مجلهٔ علمی- پژوهشی
«مهندسی برق مدرس»
دورهٔ دهم، شمارهٔ ۱، بهار ۱۳۸۹

رویکردی مبتنی بر توصیف برای تولید قوانین درستی‌یابی نرم‌افزارهای واکنشی

سید مرتضی بابامیر[1*]، سعید جلیلی[2]

۱- استادیار گروه مهندسی کامپیوتر، دانشگاه کاشان
۲- استادیار گروه مهندسی کامپیوتر، دانشگاه تربیت مدرس

***کاشان، کدپستی ۸۷۳۱۷-۵۱۱۶۷**
babamir@kashanu.ac.ir

(دریافت مقاله: تیر ۱۳۸۵، پذیرش مقاله: اسفند ۱۳۸۸)

چکیده- از آنجا که رویکردهای درستی‌یابی ایستا[1] و آزمون نرم‌افزار، برای اطمینان یافتن از درستی عملکرد نرم‌افزارها کافی نیست، رویکرد دیگری به نام درستی‌یابی در زمان اجرا[2] - که در آن درستی‌نرم‌افزار در برابر قیود[3] در زمان اجرای واقعی نرم‌افزار انجام می‌شود - مورد استقبال قرار گرفته است. اما مشکلی که این رویکرد با آن روبه‌رو است، درستی‌یابی فعالیت‌های زمان اجرای نرم‌افزار در برابر توصیف‌های انتزاعی[4] و سطح بالای قیود است زیرا ماهیت فعالیت‌های زمان‌اجرا و توصیف‌های انتزاعی با یکدیگر متفاوت است. در این مقاله با تکیه بر نرم‌افزارهای واکنشی[5]، رویکردی به نام SRG[6] ارائه می‌شود که در طی سه مرحله، از روی توصیف‌های انتزاعی مسأله و قیود، به تولید خودکار قوانین حقیقی - که برحسب کمیت‌های فعالیت‌های زمان اجرا بیان می‌شود -می‌پردازد تا درستی‌یابی رفتار اجرایی نرم‌افزار را ممکن سازد. در این رویکرد: (۱) یک مدل بصری واکنشی از توصیف مسأله ارائه شده و سپس، ضوابط رفتار حین‌اجرای نرم‌افزار برحسب منطق بی‌درنگ[7] تولید می‌شود، (۲) قیودی که باید در زمان اجرا به وسیله نرم‌افزار رعایت شوند برحسب منطق بی‌درنگ توصیف می‌شود و (۳) قوانین درستی‌یابی از قیود (مورد ۲) برحسب ضوابط رفتار نرم‌افزار (مورد ۱) تولید می‌شود. در پایان رویکرد SRG را برای مسأله "پروتکل ارتباطات پیامی" به‌کار می‌بریم.

کلیدواژگان: درستی‌یابی زمان‌اجرا، تولید قوانین درستی‌یابی، نرم‌افزار واکنشی.

۱- مقدمه

نرم‌افزار، برنامه یا مجموعه‌ای از برنامه‌ها است که پیاده‌سازی توصیف یا مدلی از مسأله موردنظر را انجام می‌دهد. نرم‌افزار واکنشی، نرم‌افزاری است که در برابر رخدادهای محیطی، با تغییر حالت خود، واکنش نشان می‌دهد. بخشی از نرم‌افزارهای واکنشی، نرم‌افزارهای حساس به ایمنی[8] هستند که در آنها رفتار نادرست نرم‌افزار موجب ارضا نشدن قیود ایمنی محیطی می‌شود. قید ایمنی یعنی این که رخداد بدی نباید در محیط

1. Static Verification
2. Run-Time Verification
3. Constraints
4. Abstract Specification
5. Reactive
6. Specification-based Rules Generation
7. Real-Time Logic
8. Safety Critical

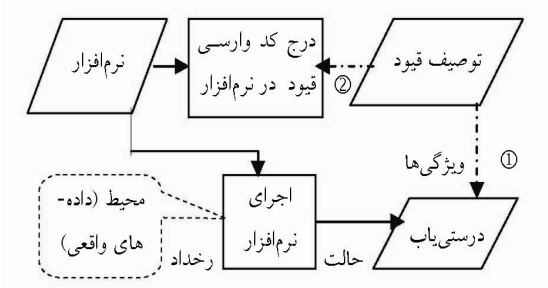
روی دهد. رخداد بد می‌تواند به فاجعه انسانی، مالی یا امنیتی منجر شود. درستی‌یابی نرم‌افزار پاسخ به این سؤال است که آیا نرم‌افزار، قیود ایمنی محیط را ارضا می‌کند؟ اگر با استفاده از روشهای درستی‌یابی توصیف –مانند اثبات قضیه یا بررسی مدل– به اثبات درستی توصیف یا مدل در برابر قیود بپردازیم، فقط ثابت کرده‌ایم که توصیف یا مدل درست است و قیود را ارضا می‌کند. بنابراین اگر نرم‌افزار، پیاده‌سازی درستی از توصیف باشد، همیشه قیود را ارضا خواهد کرد. در صورتی که از تناظر یک به یک بین توصیف و نرم‌افزار (پیاده‌سازی توصیف) مطمئن نباشیم، می‌توانیم نرم‌افزار را با داده‌های آزمایشی،بیازماییم. اما، اول آنکه اثبات قضیه و بررسی مدل در نرم‌افزارهای بزرگ و پیچیده سخت و در برخی از موارد غیرممکن است، دوم آنکه از آنجا که ممکن است مراحل بعد از توصیف مسئله یعنی طراحی و پیاده‌سازی نرم‌افزار درست و کامل نباشد (برای مثال طراحی یا پیاده‌سازی الگوریتم‌ها یا طراحی ساختارهای داده درست و کامل نباشد)، لذا نمی‌توان مطمئن بود که نرم‌افزار، پیاده‌سازی کاملا یکسانی از توصیف یا مدل است، نکته سوم این است که آزمون نرم‌افزار به دلیل انتخاب تعداد محدودی از داده‌ها و دسترسی نداشتن به تمامی اجراهای ممکن – حضور خطاها و نه غیبت آنها را نشان می‌دهد [۱] و نکته چهارم آن است که در بسیاری از موارد، شرایط محیطی که نرم‌افزار در آن اجرا می‌شود، در زمان توصیف سیستم قابل پیش‌بینی نیست.

برای مثال، نقص‌هایی که در طراحی یا پیاده‌سازی برنامه‌ها به‌وجود می‌آیند و به غیرایمن شدن آنها کمک می‌کند. مانند استفاده از اشاره‌گرهای معلق [۲] و عوامل محیطی مانند، بخشی که مترجم زبان به برنامه اضافه می‌کند. عوامل محیطی، مانند بخشهای اضافه‌شده، ممکن است نرم‌افزار را آسیب‌پذیر سازد زیرا برنامه‌نویس از آن ناآگاه است و در زمان توصیف مسأله، درستی‌یابی آن ممکن نیست. ایجاد خطاهایی مانند سرریز میانگیر[1] در فضای حافظه برنامه، ایمنی آن را به طور جدی در معرض خطر قرار دهد. در این رابطه می‌توان به نرم‌افزار LSASS[2] در سرویسگر ویندوز ۲۰۰۰ و XP اشاره کرد که با استفاده از نقص سرریز میانگیر به وسیله نفوذی‌ها تحت عنوان کاربر گمنام[3] از راه دور مورد حمله قرار گرفت[۳]. یکی از مواردی که می‌تواند موجب آسیب‌پذیری از طریق سرریز میانگیر شود، فراخوانی برنامه‌های فرعی با پارامترهای رشته‌ای در زبان‌های برنامه‌نویسی است.

بنابراین به‌منظور اطمینان از درستی نرم‌افزار، لازم است رفتار را در زمان اجرای واقعی و نه آزمایشی، نسبت به قیود درستی‌یابی کنیم. درستی‌یابی در زمان اجرا، ویژگی‌های رویکرد درستی‌یابی توصیف و رویکرد آزمون نرم‌افزار را به کار می‌برد و با محدود ساختن دامنه مسأله از ضعف‌های آنها پرهیز می‌کند. این رویکرد مانند رویکرد درستی‌یابی توصیف، به درستی‌یابی قیود، نه در توصیف مسأله بلکه در رفتار اجرای نرم‌افزار می‌پردازد. در حقیقت تفاوت اصلی رویکرد درستی‌یابی در زمان اجرا با رویکرد اثبات قضیه یا بررسی مدل آن است که به‌جای نشان‌دادن درستی تمامی اجراهای ممکن برای نرم‌افزار، فقط به درستی اجرای جاری می‌پردازد و به این ترتیب از پیچیدگی و بزرگی مسأله اجتناب می‌کند. همچنین مانند روش آزمون، با اجرای نرم‌افزار سروکار دارد اما به جای اجراهای مکرر با داده‌های آزمایشی، فقط در زمان اجرای واقعی آن را درستی‌یابی می‌کند. در این جا نیز بر خلاف روش آزمون – که با اجراهای متعدد و آزمایشی می‌خواهد تمامی رفتارهای نرم‌افزار را

1. Buffer Overflow
2 .Local Security Authority Subsystem Service
3. Anonymous User

پیش‌بینی کند- به اجرای جاری و واقعی برنامه می‌پردازد و فقط آن حالت‌هایی از نرم‌افزار را نسبت به یک یا چند قید درستی‌یابی می‌کند که در اجرای جاری برنامه پیش می‌آیند.



شکل ۱ رویکرد درستی‌یابی در زمان اجرا و شکاف بین فرآیندهای آن

شکل ۱ رویکرد درستی‌یابی در زمان اجرا را نشان می‌دهد که در آن مسیرهای بریده، چالش‌های این رویکرد، یعنی شکاف بین توصیف‌های سطح بالا و فعالیت‌های سطح پایین نرم‌افزار را مشخص می‌کند. برای پوشش‌دادن این مسیرها لازم است نگاشت بین توصیف‌های انتزاعی قیود و موجودیت‌های حقیقی زمان اجرا فراهم شود که تولید قوانین درستی‌یابی در رویکرد SRG به همین منظور است. درج قوانین درستی‌یابی در نرم‌افزار، موضوع دیگر رویکرد درستی‌یابی حین اجرا است که در این مقاله به آن نمی‌پردازیم.

رویکرد درستی‌یابی زمان اجرا که به عنوان مکمل رویکرد درستی‌یابی توصیف و رویکرد آزمون به کار می‌رود، کارایی خود را در درستی‌یابی نرم‌افزار نشان داده است. استفاده از این رویکرد برای تعیین خطاهای نرم‌افزاریی که در فرایند درستی‌یابی توصیف شده اما در آزمون نرم‌افزار کشف نشده، تاثیر به سزایی در جلوگیری از شکست نرم‌افزارهای حیاتی داشته است. برای مثال پس از درستی‌یابی توصیف و آزمون نرم‌افزار مأموریت پرواز آریان ۵، رویکرد درستی‌یابی در زمان اجرا می‌توانست از فاجعه شکست این ماموریت -که به دلیل خطای سرریز، عدد اعشاری ٦٤ بیتی را به مقدار صحیح ١٦ بیتی تبدیل کرده بود، جلوگیری کند[٤]. لازم است ذکر شود که رویکرد آزمون به تنهایی، می تواند به کشف و کاهش نرخ شکست نرم‌افزار در حدود حداکثر 10^{-4} شکست در ساعت بپردازد در حالی که در نرم‌افزارهای حساس به ایمنی نرخ قابل قبول حداکثر 10^{-9} شکست بر ساعت است [۵ و٦].

در این مقاله درستی‌یابی در زمان اجرای نرم‌افزارهای واکنشی حساس به ایمنی مد نظراست. این نرم‌افزارها مسئولیت ارضای قیود ایمنی محیط خود را به عهده دارند.

در ادامه، رویکرد SRG را در بخش دوم شرح می‌دهیم. در این رویکرد: نخست مسأله را با نگاه به واکنش نرم‌افزار به محیط، با استفاده از روشی بصری مدل‌سازی می‌کنیم. روش بصری، علاوه بر این که مسأله را به‌صورت انتزاعی و شفاف نشان می‌دهد، به فهم کاربران و متخصصان مسأله نیز نزدیک است، زیرا در آن از روابط پیچیده ریاضی و منطقی استفاده نمی‌شود. سپس مدل بصری مسأله به مدل جدولی تبدیل می‌شود که از روی این مدل، ضوابط رفتار در زمان اجرای نرم‌افزار تولید می‌شود. سپس در مرحله دوم به تعیین قیود ایمنی محیط می‌پردازیم. این قیود به‌صورت مستقیم در توصیف مسأله ظاهر نمی‌شوند، بلکه می‌توان آنها را از توصیف مسأله به‌دست آورد. معمولاً، تعداد قیود بسیار زیاد است و بنابراین باید به درستی‌یابی قیودی پرداخت که نگرانی بیشتری را برای کاربران ایجاد می کنند. اگر π پیاده‌سازی (برنامه یا نرم‌افزار) از اصول موضوعه مسأله و SAx

مجموعه‌ای از n قید باشد، آنگاه قانون درستی‌یابی را به‌صورت زیر نشان می‌دهیم:

$$Vr =^{def} \forall(sax_{i:1,n} \in SAx):\ \pi_{states} ==> sax_i$$

این قانون می‌گوید مجموعه حالاتهای نرم‌افزار در اجرا (یعنی π_{states}) باید قیود ایمنی(sax_i) را ارضا کند، در مرحله سوم قوانین درستی‌یابی رفتار نرم‌افزار را به‌دست می‌آوریم.

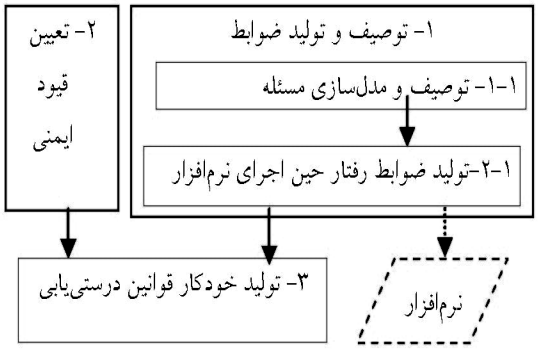
در بخش سوم مقاله، پروتکل ارتباطات پیامی را مطرح و رویکرد SRG را برای آن به کار می‌بریم. در نهایت در بخش چهارم نتیجه‌گیری آورده می شود.

۲- رویکرد SRG

در رویکرد SRG، قیود ایمنی را در سطح توصیف مسأله و با زبان کاربران آغاز می‌کنیم تا انتزاع آنها را در بالاترین سطح پشتیبانی کنیم. سپس آنها را تا سطح پیاده‌سازی دنبال می‌کنیم. بنابراین در رویکرد SRG، به‌جای درستی‌یابی مستقیم رفتار نرم‌افزار در برابر قیود، ابتدا از قیود و با توجه به ضوابط اجرایی نرم‌افزار -که از توصیف مسأله بدست می‌آید- قوانین درستی‌یابی را برحسب کمیت‌های نرم‌افزار به‌دست می‌آوریم و سپس از این قوانین برای درستی‌یابی رفتار نرم‌افزار استفاده می‌کنیم. مزیت این روش آن است که علاوه بر این که به خودکارسازی درستی‌یابی نرم‌افزار کمک می‌کند، درستی‌یابی قیود ایمنی را نیز در پیاده‌سازی اطمینان می‌دهد. در حالی که در روش مستقیم و غیرخودکار نمی‌توان از این موضوع مطمئن بود.

رویکرد SRG، سه مرحله دارد (شکل ۲). در مرحله اول، به مدل‌سازی بصری مسأله می‌پردازیم که نسبت به روشهای رسمی متنی، به فهم کاربران و متخصصان مسأله نزدیک‌تر است. سپس ضوابط رفتار زمان اجرای نرم‌افزار را از این مدل تولید می‌کنیم. در مرحله دوم، قیود ایمنی که باید نرم‌افزار آنها را در زمان اجرا ارضا کند، تعیین می‌کنیم. در پایان و در مرحله سوم، از روی توصیف قیود ایمنی و با توصیف نرم‌افزار، به تولید خودکار قوانین درستی‌یابی می‌پردازیم.

در مقایسه با رویکرد اثبات قضیه، رویکرد SRG درستی را -نه در توصیف مسأله- بلکه در پیاده‌سازی آن (یعنی نرم‌افزار) بررسی می‌کند و قضایای مورد اثبات، قیود ایمنی هستند که باید در هنگام اجرای نرم‌افزار بررسی شوند. این بررسی به دو روش انجام می‌شود.



شکل ۲ رویکرد SRG برای تولید قوانین درستی‌یابی رفتار نرم‌افزار

اول، قوانین درستی‌یابی در نرم‌افزار مقصد درج شوند تا رفتار نرم‌افزار در حین اجرای آن درستی‌یابی شود و دوم، قوانین درستی‌یابی در قالب برنامه ای جداگانه پیاده‌سازی و سپس اجرای این برنامه موازی با نرم‌افزار اصلی، اجرا شود. در این روش لازم است نرم‌افزار مقصد برنامه درستی‌یاب را فراخوانی و اطلاعات زمان اجرا را برای آن ارسال کند. روش اول برای درستی‌یابی در زمان اجرای نرم‌افزارهایی مانند نرم‌افزارهای بی‌درنگ -که در آنها زمان پاسخ نرم‌افزار به محیط اهمیت دارد- از روش دوم مناسب‌تر است. در روش دوم لازم است تا نرم‌افزار مقصد به‌صورت موقت متوقف

شود و برنامه درستی‌یاب اجراشود. در این مقاله، رویکردی را برای تولید قوانین درستی‌یابی در زمان اجرای نرم‌افزار ارائه می‌کنیم و به پیاده‌سازی قوانین و درج آنها در نرم‌افزار نمی‌پردازیم.

۲-۱- توصیف و مدل‌سازی مسأله

مدل واکنشی بر حسب روابط بین کمیت‌های پایشی (کمیت‌هایی از محیط که نرم‌افزار آنها را پایش می‌کند) و کمیت‌های کنترلی(کمیت‌هایی که نرم‌افزار به‌وسیله آنها محیط را کنترل می‌کند)، توصیف می‌شود. هدف از این بخش، توصیف مسأله بر اساس نوعی مدل واکنشی است که مراحل آن را در الگوریتم (۱) نشان داده‌ایم. پیچیدگی اجرای مراحل ۱ و ۲ به‌ترتیب برابر تعداد کمیت‌های پایشی ($|\lambda_{set}|=m$) و تعداد کمیت‌های کنترلی ($|\gamma_{set}|=c$) است که λ_{set} مجموعه کمیت‌های پایشی و γ_{set} مجموعه کمیت‌های کنترلی است. پیچیدگی تعیین روابط

۱- تعیین کمیت‌های پایشی محیط (λ_{set}) که به وسیله نرم‌افزار پایش می‌شوند، تعیین مقادیر مرزی این کمیت‌ها و ثبت زمان گذار آنها از نقاط مرزی (وقوع رخداد)،
۲- تعیین کمیت‌های کنترلی (γ_{set}) و تعیین مقادیر درست قابل اعمال روی آنها (کنترل محیط) به وسیله نرم‌افزار،
۳- تعیین روابط بین کمیت‌های پایشی و کنترلی ($\rho:\lambda_{set} \rightarrow \gamma_{set}$)

الگوریتم ۱ توصیف مسأله برحسب کمیت‌های پایشی و کنترلی

برابر تعداد رابطه‌هایی است که بین کمیت‌های محیطی و کمیت‌های کنترلی وجود دارد و حداکثر مقدار آن برابر است با $|\lambda_{set}\times\gamma_{set}|=mc$. تغییر در هر کمیت پایشی را یک رخداد در نظر می‌گیریم که نرم‌افزار باید پس از اطلاع از آن، مقدار کمیت (های) کنترلی را به‌صورت مناسب تعیین کرده و با این کار، به محیط پاسخ درست و قابل‌انتظاری بدهد. جدول (۱) نمادهای مورد استفاده را در توصیف مسأله نشان می‌دهد.

۲-۱-۱- تبیین واکنش نرم‌افزار به محیط

در مرحله اول در رویکرد SRG، مدل بصری و رسمی را از واکنش نرم‌افزار به محیط برحسب کمیت‌های پایشی و کنترلی (جدول ۱) ارائه می‌کنیم که نسبت به روشهای رسمی متنی، به فهم کاربران و متخصصین در حوزه مسأله نزدیک‌تر است. سپس شکل جدولی مدل را به‌صورت خودکار تولید می‌کنیم تا از روی آن بتوانیم ضوابط رفتار را در زمان اجرای نرم‌فزار تولید کنیم. این مجموعه ضوابط که پل ارتباطی بین توصیف و پیاده‌سازی مسأله (نرم‌افزار) است، هم مرجعی برای ساخت نرم‌افزار است و هم در درستی‌یابی قیود ایمنی استفاده می‌شوند.

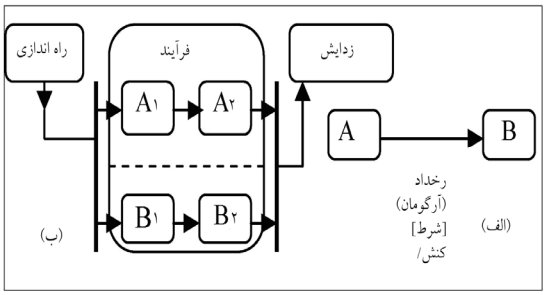
جدول ۱ کمیت‌های تعاملی محیط و نرم‌افزار

ردیف	کمیت یا رابطه	شرح
۱	λ_{set}	مجموعه کمیت‌های پایشی
۲	γ_{set}	مجموعه کمیت‌های کنترلی
۳	$\varepsilon: (\lambda_{set} \cup \gamma_{set})$	محیط(مجموعه کمیت‌های پایشی و کنترلی)
٤	$\rho:\lambda_{set} \rightarrow \gamma_{set}$	رابطه بین کمیت‌های پایشی و کنترلی
٥	$\nu:\uparrow\lambda$	رخداد آغاز یک کنش در محیط (تغییر مقدار یک کمیت پایشی از نادرست به درست)
٦	$\nu:\downarrow\lambda$	رخداد پایان یک کنش در محیط (تغییر مقدار یک کمیت پایشی از درست به نادرست)
۷	$\xi: \lambda = t$	شرط برقراری یک کمیت پایشی
۸	$\xi: \lambda = f$	شرط عدم برقراری یک کمیت پایشی
۹	$I: (\nu \cup \xi_{set})$	ورودی از محیط(یک رخداد و تعدادی شرط)

۲-۱-۲- تبیین بصری واکنش نرم‌افزار

در میان روشهای رسمی مدل‌سازی، روشهای بصری علاوه بر این که مسأله را به‌صورت انتزاعی و شفاف نشان می‌دهند، نسبت به روشهای رسمی متنی، به فهم کاربران و متخصصان مسأله نزدیک‌ترند، زیرا در آنها از روشهای

پیچیده ریاضی و منطقی استفاده نمی‌شود. روشهای مرسوم مدل‌سازی بصری عبارتند از: (۱) ماشین حالت انتزاعی[1] [۷] برای توصیف عملیاتی سیستم‌ها و یکپارچه‌سازی مدل و تحلیل دامنه مسأله با مراحل ساخت سیستم، (۲) نمودارهای توالی زنده[2] [۸] با توانایی تفکیک محیط از سیستم، نمایش تبادل‌های پیامی سیستم با محیط و نمایش رفتار اجباری (ایمنی) و ممکن (حیات) سیستم، (۳) ماشین حالت زمانی[3] با توانایی نمایش مفهوم زمان در گذارها برای سیستم‌های بی‌درنگ [۹]، (٤) ماشین حالت با گذارهای برچسب‌دار[4] برای تفکیک عملیات قابل رویت و بیرونی نرم‌افزار در برابر محیط از عملیات درونی و پنهان [۱۰]، شبکه‌های پتری و گسترش‌های آن [۱۱] با توانایی نمایش همروندی، انتخاب اولویت در همروندی، حالات زماندار و گذارهای زماندار و (۵) ماشین حالت Harel (شکل ۳) با توانایی‌های نمایش حالت‌های تودرتو، سلسله‌مراتبی و همروند [۱۲].نمودارهای حالت، نوع گسترش یافته ماشین Harel است که برای نمایش رفتار شی‌ها و تعامل بین آنها به‌کار می‌رود و به‌وسیله زبان UML پشتیبانی می‌شود [۱۳]. با توجه به توانایی‌های این ماشین حالت در ارائه رفتار واکنشی سیستم‌ها، ما در مدل‌سازی مسأله از آن بهره می‌گیریم و کمیت‌های تعامل محیط و نرم‌افزار را (جدول ۱) در آن بیان می‌کنیم (جدول ۲). شکل ۳ الف ماشین حالت Harel نمونه را در زبان UML نشان می‌دهد که در آن حالت process حالت‌های داخلی A۱، A۲، B۱ و B۲ دارد و دو حالت A۱ و A۲ و دو حالت B۱ و B۲ ترتیبی و حالت‌های A۱ و A۲ با حالت‌های B۱ و B۲ همروند هستند. پس از حالت راه‌اندازی، مسیر حالت‌های به دو شاخه تقسیم شده و قبل از حالت زدایش به یکدیگر متصل می‌شوند. شکل ۳ - ب گذار بین دو حالت را نشان می‌دهد که در آن با وقوع یک رخداد و برقراری مجموعه‌ای از شروط، ماشین از حالت A به حالت B گذار کرده و کنشی را انجام می‌دهد.



شکل ۳ الف) نمایش یک ماشین حالت Harel و **ب)** ترکیب رخداد و شرط

۲-۱-۳- مدل‌سازی جدولی

برای آن که بتوانیم ضوابط رفتار زمان اجرای نرم‌افزار را که می‌تواند برای ساخت نرم‌افزار نیز استفاده شود، به‌صورت خودکار تولید کنیم، مدل بصری مسأله را به مدل جدولی تبدیل کرده و با ارائه روشی به تولید ضوابط از روی آن می‌پردازیم.

مدل جدولی باید تا حد ممکن، توصیف کننده مدل بصری باشد، به‌صورتی که بتوان تناظری یک به یک را بین دو مدل را برقرارکرد. در این صورت تولید خودکار مدل جدولی امکان پذیر خواهد شد. از آن جا که هر گذار در ماشین حالت با یک رخداد و برقراری تعدادی شرط فعال می‌شود و انتقال حالت به وجود می‌آید، مدل جدولی باید بتواند در مقابل رخدادها تصمیم گرفته و حالت مبدأ را به‌عنوان یکی از شروط در نظر بگیرد. در حقیقت می‌خواهیم گذارهای ماشین حالت را

1. Abstract State Machine
2. Live Sequence Charts (LSCs)
3. Timed Automata
4. Labeled Transition System (LTS)

به شکل مبتنی بر رخداد توصیف کنیم به طوری که حالت مبدا هر گذار را به‌عنوان یک شرط در نظر بگیرد. به این منظور می‌توان از قواعد ECA[1] [۱٤] و روش جدولی SCR[2] [۱۵ و ۱٦] استفاده کرد. قواعد ECA در پایگاه داده‌های فعال[3] و سیستم‌های مبتنی بر قانون[4] به کار رود و پایگاه داده یا سیستم را از حالت منفعل به حالت فعال در آورده و در برابر رخدادها، کنش‌های تعیین شده را انجام می‌دهد. روش SCR با ارائه جدول‌های گذار، رخداد و شرط، ماشین حالت را از منظر گذارها، رخدادها و شروط، به‌صورت جدولی نمایش می‌دهد. ما در رویکرد SRG از روش SCR بهره می‌گیریم زیرا با جداسازی حالت‌ها از شروط، تناظر بهتری را نسبت به ECA برای ماشین حالت می‌توان نشان داد. علاوه بر این، تولید قوانین ساخت نرم‌افزار از روی آن به‌صورت الگوریتمی – با توجه به ماهیت جدولی آن – آسانتر است.

جدول ۲ کمیت‌های تعامل محیط و نرم‌افزار در ماشین حالت Harel

کمیت یا رابطه(جدول ۱)	پارامتر در ماشین حالت Harel
λ_{set}	مجموعه رخدادها
γ_{set}	مجموعه کنش‌ها
$\rho:\lambda_{set} \rightarrow \gamma_{set}$	مجموعه گذارها
$\nu:\uparrow\lambda$ / $\nu:\downarrow\lambda$	یک رخداد
$\xi:\lambda=t$ / $\lambda=f$	یک شرط
$I: (\nu \cup \xi_{set})$	یک ورودی از محیط: یک رخداد و تعدادی شرط

1. Event-Condition-Action
2. Software Cost Reduction
3. Active Database
4. Rule-Based

برای تولید خودکار ضوابط رفتار زمان اجرای نرم‌افزار، ماشین حالت Harel را به جدول حالت تبدیل می‌کنیم تا از روی آن به تولید ضوابط بپردازیم. اگر بتوان تناظری یک به یک را بین مدل بصری و مدل جدولی داشت –به شکلی که مدل جدولی نشان دهنده مستقیم مدل بصری باشد– آنگاه تولید خودکار مدل جدولی میسر خواهد شد. به این منظور، از روش جدولی SCR استفاده می‌کنیم. این روش با ارائه سه جدول (۱) گذار، (۲) رخداد و (۳) شرط به توصیف تعامل نرم‌افزار و محیط می‌پردازد [۱٦و۱۷]. هنگامی که رخدادی در محیط روی می‌دهد، نرم‌افزار با ارائه عکس‌العمل مناسبی در برابر آن، به کنترل می‌پردازد. روش SCR که در آزمایشگاه‌های تحقیقاتی نیروی دریایی امریکا تدوین شده، برای توصیف سیستم‌های کنترلی و مبتنی بر رخداد مانند سیستم الکترونیک هواپیما و کنترل روشنایی ساختمان و همچنین در توصیف پروژه‌های برخی سازمانهای صنعتی مانند آزمایشگاه‌های Bell به کار رفته است. برای توصیف سیستم به روش SCR، کمیت‌های محیطی که بر رفتار سیستم تأثیر می‌گذارند، تعیین و هر کمیت با متغیری ریاضی توصیف می‌شود.

در روش SCR برای هر متغیر محیطی که به وسیله سیستم پایش و مراقبت می‌شود، یک کلاس انتزاعی به نام "کلاس حالت" تعریف می‌شود. هر کلاس حالت معرف مجموعه حالت‌هایی است که متغیر پایشی می‌تواند داشته باشد. برای هر کلاس حالت، جدول‌های گذار و رخداد تعریف می‌شوند که هر جدول، یک تابع ریاضی را به‌صورت زیر تعریف می‌کند:

جدول‌گذار: حالت×رخداد ⟵ حالت
جدول‌گذار: حالت×رخداد ⟵ خروجی

این توابع ریاضی را که به‌صورت جدول بیان می‌شوند، به‌صورت الگوریتمی از روی ماشین حالت Harel تولید

می‌کنیم (الگوریتم‌های ۲ و ۳). فرض کنید گراف حالت (ورودی این الگوریتم‌ها) به‌صورت زوج‌های $i,j(e,\sum c_m)/a$ مشخص شوند که معرف گذارهای بین هر دو حالت i و j است و e یک رخداد، $\sum c_m$ مجموعه‌ای از شروط و a کنش وابسته به آن گذار است. در هر الگوریتم برای هر شاخص i، یک ردیف از جدول مربوطه را تشکیل می‌دهیم. الگوریتم (۲) دو حلقه تودرتو دارد که پیچیدگی آن به تعداد گذارهای ماشین حالت بستگی دارد. در بدترین حالت، هر حالت i به i-1 حالت بعدی آن متصل است که در این صورت پیچیدگی الگوریتم عبارت است از:

```
∀i: (i∈[1,n], تعدادحالت‌ها است ,n) loop
  ∀j: (j∈[i,n], n is number of states) loop
     بردار i,j(e,∑cm)/a ;
       Currentmode = mi;  Newmode = mi,j;
       ei,j = e;
  end loopj
end loopi
```

الگوریتم ۲ تولید جدول گذار از ماشین حالت Harel

```
∀i: (i∈[1,n], تعدادحالت‌ها است ,n) loop
   بردار i,j(e,∑cm)/a
     if a  = null then break loop
     Mode = mi;   ei,j = e;
     if a is high then   ri = true
                 else ri = false
end loop
```

الگوریتم ۳ تولید جدول رخداد از ماشین حالت Harel

$$(n-1)+(n-2)+\ldots+1=\frac{n(n-1)}{2}$$

تعداد پرانتزها (یعنی تعداد دور حلقه بیرونی) برابر n یعنی تعداد حالت‌های ماشین حالت و مقدار داخل هر پرانتز، تعداد حالت‌های بعدی متصل (یعنی تعداد دور حلقه داخلی) است. در حالت معمولی، می‌توان فرض کرد که هر حالت به‌طور متوسط به α حالت متصل است که α عددی ثابت است. در این حالت پیچیدگی الگوریتم ۲ برابر αn است. اما در الگوریتم ۳ فقط یک حلقه وجود دارد، لذا پیچیدگی آن از مرتبه n است.

در جدول گذار (جدول ۳)، هر حالت جدید تابعی از حالت کنونی و رخداد و در جدول رخداد (جدول ٤) هر متغیر خروجی (ردیف آخر جدول، یعنی r_i) نیز تابعی از حالت و رخداد است. جدول ۳، n حالت را نشان می‌دهد

جدول ۳ جدول گذار در SCR [۱٦]

مد جاری	رخداد شرطی	مد جدید
m_1	$e_{1,1}$ $e_{1,2}$ ... $e_{1,k1}$	$m_{1,1}$ $m_{1,2}$ ... $m_{1,k1}$
...	...	...
m_n	$e_{n,1}$ $e_{n,2}$ ... e_{n,k_n}	$m_{n,1}$ $m_{n,2}$ ... m_{n,k_n}

جدول ٤ جدول رخداد در SCR [۱٦]

مدها	رخدادهای شرطی			
m_1 m_2 ... m_n	$e_{1,1}$ $e_{2,1}$ ... $e_{n,1}$	$e_{1,2}$ $e_{2,2}$ ... $e_{n,2}$	... ...	$e_{1,p}$ $e_{2,p}$ ... $e_{n,p}$
r_i	v_1	v_2	...	v_p

که هر حالت m_i می‌تواند k_j حالت بعدی داشته باشد و جدول (٤) نیز n حالت را نشان می‌دهد که هر حالت m_k می‌تواند تحت p رخداد مختلف، p مقدار متفاوت را به متغیر خروجی r_i نسبت دهد. متغیر خروجی، متغیری است که سیستم آن را تغییر می‌دهد تا محیط را کنترل کند. برای مثال،

جدول ٤ نشان می‌دهد که اگر حالت جاری m_1 باشد و رخداد $e_{1,1}$ رخ دهد، مقدار متغیر خروجی r_i باید برابر با v_1 شود. هر رخداد در هر دو جدول می‌تواند با تعدادی شرط منطقی همراه باشد که به‌صورت true یا fasle بیان می‌شوند.

۲-۲- تولید ضوابط رفتارزمان اجرای نرم‌افزار

در این بخش از روی توصیف جدولی، ضوابط رفتار زمان اجرای نرم‌افزار را – که برای ساخت نرم‌افزار نیز استفاده می‌شود – به‌صورت خودکار تولید می‌کنیم تا با توجه به این ضوابط بتوانیم قوانین درستی‌یابی نرم‌افزار را از روی قیود ایمنی به‌صورت خودکار تولید کنیم. ضوابطی که تولید می‌شود سه نوع است: (۱) ضوابط حالت، (۲) ضوابط گذار و (۳) ضوابط پاسخ. ضوابط حالت، حالت‌های همیشه برقرار نرم‌افزار، ضوابط گذار، گذارهای همیشه برقرار بین حالت‌های نرم‌افزار، و ضوابط پاسخ، واکنش نرم‌افزار را در برابر رخدادهای محیطی نشان می‌دهند. حالت همیشه برقرار، حالتی است که نرم‌افزار باید تحت رخداد و شروط مشخصی در این حالت قرارگیرد. گذار همیشه برقرار، گذاری است که همیشه گذار بین دو حالت معینی را انجام می‌دهد. مجموعه ضوابط را می‌توان برحسب یکی از منطق‌های زمانی مانند منطق زمانی خطی[1] [۱۷]، منطق بی‌درنگ [۱۸]، حساب رخداد[2] [۱۹] و منطق بازه‌ای[3] [۲۰ و۲۱] بیان کرد. هر منطق، ویژگی‌هایی دارد که آن را برای سیستم‌های خاصی مناسب می‌سازد. برای مثال منطق بی‌درنگ و منطق بازه‌ای به‌ترتیب برای توصیف سیستم‌های بی‌درنگ و بی‌درنگ بازه‌ای و لحظه‌ای مناسب است.

ضوابط حالت، گذار و پاسخ را در رویکرد SRG براساس منطق بی‌درنگ تولید می‌کنیم زیرا این منطق در توصیف مدل‌های واکنشی و مبتنی بر رخداد کارمداست و در [۱۸] گسترشی از آن به نام نمودار مد را برای توصیف حالات سیستم که به دلیل رخدادها تغییر می‌کنند، به کار رفته است. از آن جا که مدل جدولی تولید شده، مدل بصری (یعنی ماشین حالت) را بر اساس رخدادها و حالت‌ها توصیف می‌کند، منطق بی‌درنگ و گسترش آن روش مناسبی برای نمایش ضوابط توصیف نرم‌افزار از روی مدل جدولی است. همچنین منطق بی‌درنگ برای توصیف رخدادهای لحظه‌ای برحسب زمان مطلق و گسترش آن برای نشان‌دادن دوره زمانی حالت‌های قابل استفاده هستند. در نتیجه می‌توان از منطق بی‌درنگ و گسترش آن، همچنین برای توصیف مدل‌های واکنشی و مبتنی بر رخداد و هم برای توصیف حالت‌های سیستم که به دلیل رخدادها تغییر می‌کنند، استفاده کرد. بنابراین از آن جا که مدل جدولی، نوعی توصیف مبتنی بر رخداد را ارائه می‌کند، منطق بی‌درنگ روش مناسبی برای نمایش قوانین ساخت نرم‌افزار از روی مدل جدولی است.

منطق بی‌درنگ، نوعی منطق زمانی گسسته، دارای ساعت و مبتنی بر رخداد با مهر زمانی است. این منطق علاوه بر ترتیب تقدم و تأخر بین رخدادها، زمان مطلق و نسبی را نیز توصیف می‌کند. منطق بی‌درنگ هم برای تحلیل ایستا و هم برای تحلیل پویای سیستم‌های بی‌درنگ به کار می رود[۱۸]. در این منطق، رخداد لحظه ای زمانی است و عمل، دوره‌ای از زمان را مشخص می‌کند که در آن کنشی انجام می‌شود و راه‌اندازی کنش با یک رخداد و تکمیل آن با رخدادی دیگر تعیین می‌شود. چهار کلاس رخداد وجود دارد[۲۲]: (۱) رخداد خارجی(نماد Ω)، (۲) رخداد شروع(نماد ↑) و (۳)رخداد توقف (نماد↓) که به‌ترتیب آغاز و تکمیل کنش را مشخص می‌کنند و (٤) رخداد گذار حالت که تغییر یک ویژگی خاص سیستم را مشخص می‌کند. در این منطق، زمان رخداد و مقدار متغیر به‌ترتیب با رابطه‌های زیر مشخص می‌شوند:

1. Linear Temporal Logic
2. Event Calculus
3. Interval Logic

زمان ← دفعه رخداد× واقعه:@(e,i)

مقدار ← پیدایش × متغیر: @val(v,i)

occurrence، عددی صحیح و مثبت (معرف i امین رخداد یا مقدار بعدی نسبت به زمان جاری)، یا عددی صحیح و منفی (معرف i امین رخداد یا مقدار "اخیر" در گذشته نسبت به زمان جاری) است و مقدار صفر برای آن تعریف نمی شود. جدول ۵ معنای توابع و قیود بین رخدادها را که در منطق بی‌درنگ به کار می‌روند، نشان می‌دهد. هر قید به‌صورت C) @ (e,i) ≤ @ (f,j) ±C عددی صحیح) بیان می‌شود و "مهلت" است اگر C مثبت و "تاخیر" است اگر C منفی باشد. حالت‌های سیستم به وسیله نمودار مد و با یک بازه تعریف می‌شوند که ورود به هر حالت یا خروج از آن، یک رخداد است. جدول (٦) رخدادهای مرتبط با حالت M را نشان می‌دهد.

جدول ۵ توابع و قیود بین رخدادها در منطق بی‌درنگ [۱۸]

قید یا تابع	معنا
t=@(e,i)	تابعی که زمان i امین رخداد e را می‌دهد
t=@(e,-1)	تابعی که زمان آخرین رخداد e را می‌دهد
u=@val(v,i)	تابعی که مقدار i امین متغیر v را می‌دهد
@(e1,i) ≤ @(e2,i) + 20	قید مهلت (i امین رخداد واقعه e_1 باید حداکثر در ۲۰ واحد زمانی پس از i امین رخداد واقعه e_2 رخ دهد)
@(e,i) +10 ≤ @(e,i+1)	قید تاخیر (حداقل باید ۱۰ واحد زمانی بین دو رخداد متوالی e فاصله باشد)
@(e,i+5) ≥ @(e,i) +50	قید مهلت (رخداد e حداقل باید ۵ بار در هر ۵۰ واحد زمانی رخ دهد)
@(Ωe, i) < @(↑A,i) ∧ @(↓A,i)≤ @(Ωe,i) + 20	کنش A پس از رخداد خارجی e شروع و پس از گذشت ۲۰ ثانیه از آن باید کامل شود

جدول ٦ رخدادهای مرتبط با حالت در نمودار مد [۱۹]

رخداد	معنا
$M[t_1,t_2]$	سیستم در زمان t_1 به حالت M وارد و در زمان t_2 از آن خارج می‌شود.
$M[t_1,t_2)$	سیستم در زمان t_1 به حالت M وارد و در زمان t_2 یا بعد از آن از آن خارج می‌شود.
$M[t_1,t_2>$	سیستم در زمان t_1 به حالت M وارد و در زمانی بعد از t_2 از آن خارج می‌شود
$M(t_1,t_2)$	سیستم در زمان t_1 یا قبل از آن به حالت M وارد و در زمان t_2 یا بعد از آن از آن خارج می‌شود
$M<t_1, t_2>$	سیستم قبل از زمان t_1 به حالت M وارد و پس از زمان t_2 از آن خارج می‌شود

برای نشان‌دادن: (۱) بروز رخدادی در نقطه زمانی خاص که به معنای تغییر مقدار کمیتی درگذار از حالت جاری به حالت جدید است، (۲) شرط که به معنای تغییر نکردن مقدار کمیت در گذار از حالت جاری به حالت جدید و (۳) ورود به حالت و خروج از آن، نمادهایی را برحسب روابط منطق بی‌درنگ در جدول (۷) مشخص می‌کنیم. این جدول، رخدادها و شروط مرتبط با کمیت‌های پایشی را برحسب فرمول‌های منطق بی‌درنگ نشان می‌دهد که در آن $\uparrow\lambda t$، $\downarrow\lambda t$، $\searrow St$ و $\nearrow St$ معرف کمیت v و $\bar{\tau}\lambda t$ و $\pm\lambda t$ معرف کمیت ξ از جدول ۱ است. @T(λ) و @F(λ) به ترتیب به معنی "برقراری" و "عدم برقراری" کمیت λ است.

۲-۲-۱- تولید ضوابط حالت

ضوابط حالت، حالت‌های همیشه برقرار را باید مشخص کنند. از آن جا که نرم‌افزار در هر زمان فقط در یکی از حالت‌های خویش قرار دارد، اولین ضابطه حالت باید انحصار حالت‌های را نشان دهد که ما آن را در ضابطه a-۱ به‌صورت کلی و در ضابطه b-۱ با استفاده از نمادهای جدول ٦ برحسب منطق بی‌درنگ مشخص کرده‌ایم. بقیه

ضوابط حالت باید وضعیت‌های همیشه برقرار مختلف را نشان دهند. این ضوابط را با توجه به رخدادهایی که در هر حالت رخ می‌دهند و شروطی که در آن حالت برقرارند (یعنی رخدادها شرطی هستند) به‌وسیله الگوریتم

جدول ۷ تعریف نمادها در SRG با گزاره‌های منطق بی‌درنگ

نماد	گزاره	شرح
$\uparrow\lambda_t$	$@val(\lambda,k) = false \wedge \exists e=@T(\lambda) \therefore t=@(e,k+1)$	آغاز کنش در زمان t (مقدار λ در زمان t برابر true می‌شود)
$\downarrow\lambda_t$	$@val(\lambda,k) = true \wedge \exists e=@F(\lambda) \therefore t=@(e,k+1)$	پایان کنش (مقدار λ در زمان t برابر false می‌شود)
$\nearrow S_t$	S(t-1,t]	پایان حالت (در زمان t از حالت S خارج می‌شویم)
$\searrow S_t$	S[t,t+1)	شروع حالت (در زمان t به حالت S وارد می‌شویم)
$\leftrightarrows S_t$	S(t,t)	در نقطه زمانی t در حالت S قرار داریم
$\leftrightarrows S_{t1,t2}$	$S[t_1,t_2]$	در بازه زمانی t_1 تا t_2 در حالت S قرار داریم
$\bar{\uparrow}\lambda_t$	$!\exists e=@T(\lambda) \therefore t=@(e,k) \wedge @val(\lambda,k) = false$	کنش نه در زمان t شروع می‌شود و نه در این زمان ادامه دارد
$\bar{\downarrow}\lambda_t$	$!\exists e=@F(\lambda) \therefore t=@(e,k) \wedge @val(\lambda,k) = true$	کنش در زمان t ادامه دارد

$MI_0(t)$ def≡ (الف)

$S = \bigcup_{i=1}^{n} State_i$, $T = \bigcup_{k=0}^{\infty} Time_k$

$\forall t_k \in T, \forall t'_k \in T, \forall State_i \in S, \forall State_j \in S,$
if $State_i(t_k,t'_k) \wedge State_j(t_k,t'_k) \Rightarrow i = j$

$MI_0(t)$ def≡ (ب)

$\leftrightarrows State_{1\ (t1,t2)} \oplus \leftrightarrows State_{2\ (t1,t2)} \oplus \ldots \oplus \leftrightarrows State_{n\ (t1,t2)}$

ضابطه ۱ (الف) ضابطه انحصار حالت‌ها در نرم‌افزار به‌صورت کلی و **(ب)** ضابطه انحصار حالت‌ها برحسب روابط منطق بی‌درنگ

(٤) در بخش (الف) به‌صورت خودکار تولید می‌کنیم. همانطور که مشاهده می‌شود، برای هر حالت یک ضابطه داریم که با MI نشان می‌دهیم. در الگوریتم (٤) با اجرای یک حلقه برای هر حالت جاری (هر ردیف) از جدول گذار حالت، وضعیت تمام کمیت‌ها (رخداد یا شرط) دریافت می‌شوند و از آنجا که این الگوریتم ورودی خود را از خروجی الگوریتم (۲) (یعنی جدول ۳) می‌گیرد، پیچیدگی آن با پیچیدگی الگوریتم (۲) یکسان است. اگر کمیت رخداد، شروع کنش باشد(یعنی ↑)، شرط false و اگر یک کمیت، رخداد خاتمه کنش باشد (یعنی ↓)، شرط true را برای حالت جاری در نظر می‌گیریم.

$S = \bigcup_{i=1}^{n} State_i$, $T = \bigcup_{k=0}^{\infty} Time_k$, $Q = \bigcup_{j=1}^{m} Quantity_j$

for i=1 to n do *(الف)*
  conds = null
  for j=1 to m do
    if q_j="↑" or q_j="f" then conds=conds ∪ false
    if q_j = "↓" or q_j="t" then conds=conds ∪ true
  end for j ($q_j \in Q$)
  $MI_{i\in 1,n}(t)$ def≡ if $State_i$ [t] → conds
end for i

(ب) داده‌های ورودی به الگوریتم

Old Mode	λ_1	λ_2	λ_3	New Mode
$State_1$	f	↑	t	$State_2$

Output:
$MI_1(t)$ def≡ ∀t if $\leftrightarrows State_{1\ [t]} \rightarrow @val(\lambda_1,-1) = false \wedge @val(\lambda_2,-1) = false \wedge @val(\lambda_3,-1) = true$

الگوریتم ٤ (الف) تولید ضوابط حالت و **(ب)** یک ضابطه نمونه حالت (خروجی الگوریتم) برحسب منطق بی‌درنگ

در بخش (ب)، داده ورودی این الگوریتم و خروجی متناظر را با استفاده از نمادگذاری جدول (٦) برحسب منطق بی‌درنگ نشان می‌دهیم که در آن،

@val(v,-1) آخرین مقدار متغیر حالت v را نشان می‌دهد. برای مثال ضابطه $MI_1(t)$ می گوید: هر گاه نرم‌افزار در $State_1$ باشد، آخرین مقدار کمیت‌های ورودی λ_1 و λ_2 برابر false و آخرین مقدار کمیت λ_3 برابر true است.

۲-۲-۲- تولید ضوابط‌گذار

ضوابط‌گذار، ضوابط گذارهای همیشه برقرار یعنی گذارهایی را که همیشه انتقال بین حالت‌های معینی را انجام می‌دهند، مشخص می‌کنند. روش تولید این ضوابط چنین است: برای هر گذار از حالت جاری به حالت جدید در جدول گذار حالت‌ها (جدول ۲) یک ضابطه برای خروج از حالت جاری و یک ضابطه برای ورود به حالت جدید تولید می‌کنیم (بخش (الف) از الگوریتم۵). کمیت‌های هر ردیف از جدول گذار حالت‌ها، فرضهای ضابطه ورود به حالت جدید و ضابطه خروج از حالت جاری را برای آن ردیف تعیین می‌کند به‌طوری که اگر مقدار کمیت، رخداد شروع کنش را نشان دهد (یعنی "↑") شرط false را برای این کمیت به ضابطه خروج و شرط true را به ضابطه ورود اضافه می کنیم و اگر رخداد خاتمهٔ کنش را نشان دهد (یعنی ↓)، شرط true را برای این کمیت به ضابطه خروج و شرط false را به ضابطه ورود اضافه می‌کنیم. به ازای هر مقدار ثابت t یا f از کمیت، مقدار true یا false به هر دو ضابطه اضافه می‌شود. بخش(ب)در الگوریتم (۵)، خروجی این الگوریتم را برای داده ورودی بخش(ب)از الگوریتم (٤) نشان می‌دهد. پیچیدگی این الگوریتم مشابه پیچیدگی الگوریتم (۳) و از مرتبه $n\times\alpha$ است.

$T = \bigcup_{k=0}^{\infty} Time_k$, $Q = \bigcup_{j=1}^{m} Quantity_j$

conds = null

$\forall row_i$ in state transition table $\wedge \forall t \in T$ do: (الف)

$\forall q_j \in Q$ in row_i do:

switch q_j

case "f" $E = \bar{f}Q_t$; break;

case "t" $E = \downarrow Q_t$; break;

case "↑" $E = \uparrow Q_t$; break;

case"↓" $E = \downarrow Q_t$;

conds = conds $\wedge$ E

end do q_j

TI_{i1} def≡ if $(\leftrightarrows OldState_t \wedge conds) \rightarrow \nearrow OldState_{t+1}$

TI_{i2} def≡ if $(\leftrightarrows NewState_t \wedge conds) \rightarrow \searrow NewState_{t+1}$

end do row_i

TI_{11} def≡ $\forall t$ if $\leftrightarrows State_{1\,[t]} \wedge (\bar{f}\lambda_{1\,t} \wedge \downarrow\lambda_{3\,t} \wedge \uparrow\lambda_{2\,t}) \rightarrow \nearrow State_{1\,[t+1]}$

TI_{12} def≡ $\forall t$ if $\leftrightarrows State_{1\,[t]} \wedge (\bar{f}\lambda_{1\,t} \wedge \downarrow\lambda_{3\,t} \wedge \uparrow\lambda_{2\,t}) \rightarrow \searrow State_{2\,[t+1]}$ (ب)

الگوریتم ۵- (الف) تولید ضوابط گذار و (ب) تولید دو ضابطه نمونه گذار بازا ورودی بخش ب از الگوریتم ٤

۲-۲-۳- تولید ضوابط پاسخ

ضوابط پاسخ، واکنش‌های همواره لازم و معین نرم‌افزار را به رخدادهای محیطی به‌وسیله تغییر کمیت‌های کنترلی نشان می‌دهد. از آنجا که در رویکرد SRG، مدل جدولی را با استفاده از روش SCR توصیف کردیم و این روش، پاسخ به رخدادهای محیطی را به‌وسیله جدول رخداد (جدول ۳) توصیف می کند، ضوابط پاسخ را به‌طورمستقیم از روی جدول‌های رخداد تولید می‌کنیم. در ردیف آخر هر جدول رخداد، یک کمیت کنترلی مشخص شده که نرم‌افزار با تغییر آن به محیط پاسخ می‌دهد. ردیف‌های دیگر جدول رخداد،حالت‌هایی است که در آن حالات، تحت رخدادهای خاصی باید پاسخ مورد نظر (یعنی تغییر مقدار کمیت) داده شود. مقدار false برای یک رخداد به معنای آن است که در آن حالت، نرم‌افزار به رخداد پاسخ نمی‌دهد. الگوریتم (٦)، تولید ضوابط پاسخ را از روی جدول رخداد نشان می‌دهد

(جدول ۳). پیچیدگی این الگوریتم مشابه الگـوریتم (۳) و از مرتبه $n\times\alpha$ است.

۲-۳- تعیین قیود ایمنی

قیود ایمنی قیودی است که نرم‌افزار (یعنی پیاده‌سازی توصیف)، باید آنها را ارضا کند. قیود می‌توانند به‌صورت مستقیم در توصیف مسأله ظاهر نشوند، بلکه با استنتاج از اصول توصیف به‌دست آیند. بنابراین می‌توان توصیف را به‌عنوان مجموعه اصول و قیود را به‌عنوان گزاره‌هایی در نظر گرفت که باید از این اصول استنتاج شوند. در حقیقت در اینجا رویکردی هدف‌گرا را دنبال می‌کنیم و مجموعه اهداف، قیود ایمنی است. این مجموعه اهداف، اطلاعات قابل استنتاج از توصیف است (اگر توصیف درست باشد). در رویکرد SRG به جای این که به درستی‌یابی قیود در توصیف بپردازیم، به دنبال ارضای آنها در رفتار اجرای جاری نرم‌افزار هستیم.

$S = \bigcup_{i=1}^{n} State_i$, $T = \bigcup_{k=0}^{\infty} Time_k$, $PQ = \bigcup_{j=1}^{m} PassiveQuantity_j$

$\forall row_i$ in event table do:
  loop: $\forall$ $event_j$ in row_i do:
    if $event_j$ = "false" then loop;
    if pq = true then
      "$RI_{ij}(t)$ def≡ if $State_i$ [t] $\wedge$ $event_j$ $\wedge \rightarrow \uparrow$pq" ;
    else
      "$RI_{ij}(t)$ def≡ if $State_i$ [t] $\wedge$ $event_j$ $\wedge \rightarrow \downarrow$pq" ;
end do row_i

الگوریتم ٦ تولید ضوابط پاسخ

هر قید ایمنی (در ضمن اجرای جاری برنامه رویداد بدی رخ نمی‌دهد) بر اساس مشاهده حالت‌های اجرای جاری برنامه که متناهی است، بررسی - می‌شود [۱۵و۲۲و۲۳]. قیود ایمنی که بخش بزرگی از ویژگی‌های نیازهای هر نرم‌افزار حساس به ایمنی را تشکیل می‌دهند، مجموعه‌ای از نبایدها برای رخدادهای مخاطره‌آمیز است، اما قیود ایمنی هر سیستم، ویژه همان سیستم است و باید به‌صورت خاص برای آن تعیین شود. شکل کلی هر قید ایمنی را به‌صورت جمله ای خبری یا شرطی برحسب نوعی رابطه در منطق بی‌درنگ توصیف می‌کنیم. در جمله خبری، عدم وقوع رخداد (مانند زمان انتظار نباید از ۱۰ ثانیه بیشتر شود)، بیان می‌شود. در جمله شرطی، در بخش فرض، ترکیبی از رخدادها و شروط، و در بخش نتیجه، نبود روابط مخاطره‌آمیز بین رخدادها و شروط بیان می‌شوند. هر رخداد به یکی از شکلهای $\uparrow\lambda$، $\downarrow\lambda$، $\nearrow S$ و $S\searrow$ و هر شرط به یکی از شکلهای $S\leftrightarrows$، $\bar{t}\lambda$ و $\pm\lambda$ مشخص می‌شود (جدول (۷)).

۲-٤- تولید خودکار قوانین درستی‌یابی

ما در بخش ۱، درستی‌یابی رفتار برنامه (π_{states}) را در برابر قیود ایمنی به‌وسیله قوانین وارسی (Vr) بیان کردیم. اما نمی‌توان بدون منظور کردن محیط (ε) و شرایطی که نرم‌افزار در آن اجرا می‌شود، به ارضای قیود ایمنی پرداخت. زیرا در این صورت محیط می‌تواند باعث شود که نرم‌افزار به‌صورت غیرقابل انتظاری عمل کند. بنابراین لازم است بررسی شود که آیا π، SAx را در ε ارضا می‌کند؟ پس اگر ε را که π در آن عمل می‌کند نیز در نظر بگیریم، قانون درستی‌یابی (Vr) به‌صورت زیر بیان می‌شود:

فرمول ۱:

$$Vr =^{def} \forall(sax_{i:1,n} \in SAx): (\pi_{states}, \varepsilon) ==> sax_i$$

این رابطه نشان می‌دهد که π در ε، SAx را ارضا می‌کند.

اکنون اگر وارسی‌کننده‌ای به نام μ را در نظر بگیریم که π را در زمان اجرا در ε پایش و مراقبت می‌کند، آنگاه

رابطه (۲) (درستی‌یابی قیود ایمنی) بیان می‌کند که اگر π در ε، SAx را ارضا نکند، پیام اخطار (η) صادر می‌شود:

فرمول ۲:

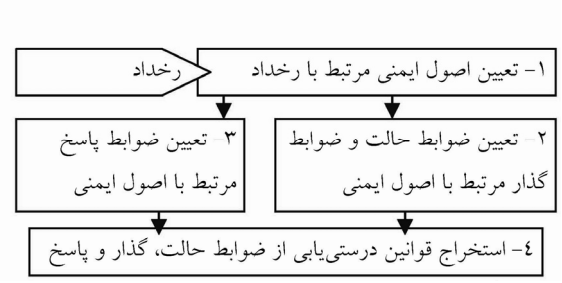
$$Vr =^{def} \forall(sax_{i:1,n} \in SAx): (\pi_{states}, \varepsilon, \mu) ==> sax_i \veebar \sim sax_i \wedge \eta$$

اما برای این که μ بتواند قید ایمنی یا عدم آن را (با η) اعلام کند، لازم است به تعامل بین π و ε پردازیم و از روی آن تناظر بین π_{states} و رخدادهای ε را تعیین کنیم زیرا μ هیچ اطلاعاتی از ε، غیر از آن چه از اعلام یا ثبت رخدادها استنباط می‌شود، نمی‌تواند به‌دست آورد.

اکنون به روش تولید خودکار Vr (قوانین درستی‌یابی) می‌پردازیم. از آنجا که در رویکرد درستی‌یابی اجرا، μ به درستی‌یابی رفتار زمان اجرای نرم‌افزار (π_{states}) در برابر قیود ایمنی می‌پردازد تا پیروی یا انحراف رفتار π را از آن تعیین کند، بنابراین برای هر رخدادی که در زمان اجرای نرم‌افزار رخ می‌دهد، باید π_{states} را در واکنش به این رخداد برای هر sax_i مرتبط درستی‌یابی کنیم. به این منظور الگوریتم (۷) را برای تولید خودکار قوانین درستی‌یابی نرم‌افزار در برابر قیود ایمنی ارائه می‌کنیم. شکل (٤) این چهار مرحله را نشان می‌دهد. پیچیدگی اجرای الگوریتم (۷) برابر است با $s+k_1+k_2$ که $|sax|=s$ پیچیدگی مرحله (۱) را در این الگوریتم نشان می‌دهد و برابر با تعداد قیود ایمنی مسأله است. k_1، پیچیدگی مرحله ۲ و k_2 پیچیدگی مرحله ۳ الگوریتم (۷) را نشان می‌دهد. حداکثر k_1 برابر با تعداد ردیف‌ها در جدول گذار و حداکثر k_2 برابر با تعداد ردیف‌ها در جدول رخداد است.

۱- رخداد و شرایط محیط(ε) را در فرض هر قید ایمنی، sax_i، بررسی می‌کنیم تا قید مرتبط را مشخص کنیم،
۲- برای هر قید مرتبط، ردیفی از جدول گذار را که متناظر با نتیجه یک قید است، بدست می‌آوریم، و ضوابط حالت و گذار متناظر با این ردیف‌ها را تعیین می‌کنیم،
۳- واکنش مطلوب نرم‌افزار را در برابر رخداد را از جدول رخداد بدست آورده و ضابطه پاسخ متناظر را تعیین می‌کنیم،
٤- قوانین درستی‌یابی را از روی ضوابط حالت، گذار و پاسخ بدست آمده در بندهای ۲ و ۳، به‌صورت خودکار تولید می‌کنیم.

الگوریتم ۷ تولید خودکار قوانین درستی‌یابی نرم‌افزار



شکل ٤ مراحل روش تولید قوانین درستی‌یابی رفتار نرم‌افزار

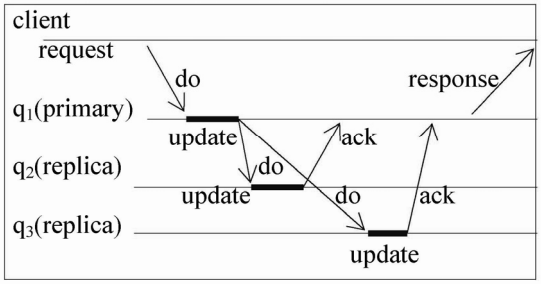
۳-طرح مسأله پروتکل ارتباطات گروهی

در سیستم‌های متقاضی / سرویسگر، استفاده از سرویسگر متمرکز و منفرد مانند پایگاه داده، ساده‌ترین راه برای ارائه خدمات است، اما به دلیل امکان بروز مشکل در سرویسگر مرکزی، این روش سرویس‌دهی غیر ایمن است و می‌تواند سیستم‌های حساس به ایمنی را ناامن سازد. روشی برای افزایش ایمنی و تحمل‌پذیری خطا، روش «پشتیبان یا همتا[1]» است[۲٤]. در این روش، یکی از سرویسگرها، نقش سرویسگر اصلی و دیگران، نقش سرویسگر پشتیبان یا همتا را ایفا می‌کنند. تقاضاها، فقط به وسیله سرویسگر اصلی پاسخ داده می‌شود و اگر این سرویسگر داده‌ای را تغییر دهد، سایر سرویسگرها را مطلع

1. Backup/Replica

می‌سازد تا آنها نیز داده متناظر را تغییر دهند و به این ترتیب سازگاری داده‌ها در سیستم رعایت می‌شود (شکل ۵). اما هنگامی که سرویسگر اصلی دچار مشکل می‌شود، یکی از همتاها، نقش سرویسگر اصلی را به عهده می‌گیرد. بنابراین در چنین سیستمی، هر همتا باید نسخه یکسان با داده‌های سرویسگر اصلی را در اختیار داشته باشد تا در موقع لزوم به ارائه خدمات بپردازد.

همانطور که شکل (۵) نشان می‌دهد، متقاضی درخواست تغییر داده را برای سرویسگر اصلی (q_1) ارسال می‌کند. سرویسگر پس از تغییر داده در مخزن خود، از دو همتای دیگر (q_2,q_3) نیز درخواست تغییر داده می‌کند. این دو همتا پس از تغییر داده، پیام تصدیق را به q_1 ارسال کرده و سپس q_1 به متقاضی پاسخ می‌دهد. برای حفظ یکپارچگی و سازگاری در روش پشتیبان و موارد مشابه دیگری که در سیستم‌های غیرمتمرکز استفاده می‌شود، به نوعی پروتکل ارتباط داده در ارتباطات گروهی[1] نیاز است. این پروتکل ارتباطی بر خلاف پروتکل‌های ارتباطی یک به یک –مانند پروتکل TCP– باید ارتباط یک به چند را پشتیبانی کند. در این پروتکل ارتباطی که چند پخشی[2] نامیده می‌شود، پیامی از یک عضو گروه (مانند سرویسگر اصلی) به تمامی اعضای گروه (مانند سرویسگرهای همتا) ارسال می‌شود.



شکل ۵ سازوکار اصلاح داده در روش پشتیبان با دو همتا

1. Group Communication System
2. Multicast

اما در سیستم‌های غیر متمرکز (مانند سیستم بانکی) که به‌دلیل افزایش تحمل‌پذیری خطا لازم است تکرار داده‌ها (مانند حساب مشتریان) را در سرویسگرهای پشتیبان داشته باشیم، استفاده از پروتکل چندپخشی برای نگاهداری یکپارچگی و سازگاری داده‌ها با دو مانع روبه‌رو است: (۱) به‌دلیل شکست یکی از همتاها یا به‌دلیل قطع ارتباط شبکه‌ای، پیام تغییر یا اصلاح داده توسط همتا دریافت نمی‌شود و در نتیجه، داده‌های همتا، قدیمی و بی‌اعتبار می‌شود و (۲) به‌دلیل مشکلات شبکه‌ای مانند ترافیک یا قطعی، پیام‌های تغییر یا اصلاح داده‌ها، به‌ترتیبِ ارسال آنها به سرویسگرهای همتا نمی‌رسد و در نتیجه، مقدار نهایی آنها در سرویسگرهای همتا متفاوت می‌شود. بنابراین باید پروتکل چندپخشی به‌صورتی قابل اطمینان شود که در صورت بروز مشکلی در یکی از سرویسگرها یا خطوط شبکه، سازگاری داده‌ها حفظ شود و در نتیجه کنش‌های سرویسگرها به‌دلیل داده‌های نادرست، ناامن نباشد (برای مثال یکی از سرویسگرهای همتا، مشتری را بدهکار و دیگری آن را بستانکار بداند).

۳-۱-توصیف پروتکل ارتباطات گروهی

در این پروتکل توزیعی، هر عضو گروه (که پیامی را ارسال و برای آن تصدیقی را دریافت می‌کند)، نقش فرستنده و سایر اعضا (که آن پیام را دریافت و تصدیق را ارسال می‌کنند)، نقش گیرنده را دارند. حالت‌های فرستنده و حالت‌های گیرنده پیام در نتیجه رخدادهای (تقاضاهای) ارسال و دریافت تغییر می‌کند (جدول‌های ۸ و ۹). هر حالت در این جدول‌ها معرف یک دوره کنش است (یعنی فرستنده یا گیرنده پیام در حال انجام عمل خاصی هستند). RDY حالت اولیه، SDG، AWT و RVG حالت‌های موقت فرستنده و گیرنده

جدول ۸ جدول حالات فرستنده پیام

نام	تعریف
RDY	فرستنده آماده ارسال پیام است
SDG	فرستنده در حال ارسال پیام است
AWT	فرستنده منتظر دریافت تصدیق است
SUC	فرستنده پیام تصدیق را در مهلت مقرر دریافت کرده است
FAL	مهلت مقرر منقضی شده است و پیام تصدیق دریافت نشده است
DED	فرستنده خراب است

جدول ۹ جدول حالات گیرنده پیام

نام	تعریف
RDY	گیرنده آماده دریافت پیام است
RVG	گیرنده در حال دریافت پیام است
SUC	گیرنده پیام را دریافت کرده است
DED	گیرنده خراب است

و SUC، FAL و DED حالت‌های نهایی هستند. اکنون بر طبق الگوریتم (۱) به تعامل اعضا گروه و تعیین کمیت‌های محیطی می‌پردازیم تا توصیف اولیه سیستم را برحسب جدول (۱) به‌دست آوریم. کمیت‌های پایشی (λ) و کنترلی (γ) (ردیف‌های ۱و۲ از جدول ۱) و رخدادهای $\lambda\uparrow$ و $\lambda\downarrow$ (ردیف ۵ و ٦ از جدول ۱) را در جدول‌های (۱۰) و (۱۱) نشان می‌دهیم. این جدول‌ها، کمیت‌های ورودی (پایشی) از محیط به فرستنده و گیرنده و کمیت‌های خروجی (کنترلی) از آنها به محیط و همچنین رخدادهای مرتبط با آنها (ستون‌های با عناوین ↑ و ↓) را نشان می‌دهد. تغییر هر کمیت (در اینجا کمیت‌ها منطقی هستند)، رخدادی است که نشان دهنده شروع یک عمل (ستون با عنوان ↑) یا پایان آن (ستون با عنوان ↓) است.

۳-۱-۱- مدل‌سازی بصری ارتباطات گروهی

همانطور که در بخش ۲-۱ توضیح دادیم، برای مدل‌سازی بصری سیستم می‌توان از انواع ماشین‌های حالت بهره گرفت. با توجه به ویژگی‌هایی که در بخش ۲-۱ برای ماشین حالت Harel بیان شد، در اینجا برای مدل‌سازی بصری ارتباطات گروهی از این ماشین حالت بهره می‌گیریم. شکل‌های (٦) و (۷) به‌ترتیب ماشین حالت را برای فرستنده و گیرنده پیام نشان می‌دهد. رخدادها با ↑ و ↓ و شروط با جفت [...] مشخص شده است.

جدول ۱۰ توصیف فرستنده پیام

کمیت	نوع	تعریف	↑	↓
Snd	λ	ارسال پیام	شروع	پایان
RAck	λ	دریافت تصدیق	شروع	پایان
Tout	λ	مهلت دریافت تصدیق	-	پایان
Crs	داخلی	خرابی فرستنده	خراب	سالم
Stm	γ	شمارش زمان به وسیله تایمر	شروع	پایان

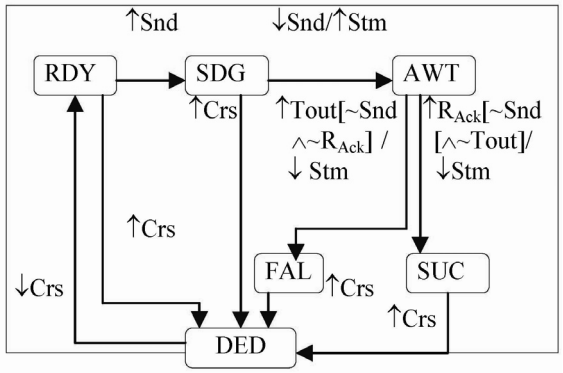
جدول ۱۱ توصیف گیرنده پیام

کمیت	نوع	تعریف	↑	↓
Rcv	λ	پردازش یک پیام	شروع	پایان
Crs	داخلی	خرابی گیرنده	خراب	سالم

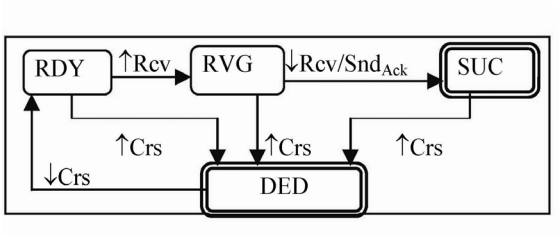
۳-۱-۲ مدل‌سازی جدولی

اکنون الگوریتم (۲) (بخش ۲-۱) را برای مدل بصری فرستنده و گیرنده پیام (شکل ٦ و شکل ۷) در ارتباطات گروهی به‌کار برده و جدول‌های گذار را تولید می‌کنیم (جدول‌های ۱۲ و ۱۳). به‌طور مشابه الگوریتم (۳) را نیز

به‌کار برده و جدول‌های رخداد (جدول‌های ١٤ و ١٥) را تولید می‌کنیم. در الگوریتم ٢ در هر دور اجرا، یک گذار بین دو حالت ماشین حالت استخراج (اگر وجود داشته باشد) و برای آن یک ردیف از جدول گذار تولید می‌شود. در الگوریتم (٣) نیز در هر دور اجرا، یک گذار بین دو حالت ماشین حالت استخراج می‌شود (اگر وجود داشته باشد) تا مقدار true یا false برای متغیر خروجی (یعنی r_i) مشخص شود (اگر گذار، متغیر خروجی داشته باشد). در این جدول‌ها، ↑ و ↓ به‌ترتیب رخداد شروع و رخداد پایان کنش محیطی را نشان می‌دهد و t و f شروط این رخداد را (یعنی رخدادها شرطی هستند). حالت‌ها (مدها) و رخدادها در جدول‌های (٨) تا (١١) شرح داده شده است. اگر در یک حالت، رخدادی برابر false باشد (در ستون رخداد از جدول رخداد) به معنای آن است که در آن حالت، هیچ رخدادی روی نمی‌دهد که بر کمیت خروجی تأثیر داشته باشد. برای مثال مقدار false در جدول (١٤) (ردیف دوم، ستون دوم)، به این معنا است که هنگامی که فرستنده در حالت AWT قرار دارد، هیچ رخدادی نمی‌تواند روی دهد که کمیت Stm را به مقدار true تغییر دهد، اما در این حالت تحت رخداد شرطی RACK↑ یا رخداد شرطی Tout↑، کمیت Stm به مقدار false تغییر می‌یابد.



شکل ٦ ماشین حالت برای فرستنده



شکل ٧ ماشین حالت برای گیرنده

جدول ١٢ جدول گذار حالت برای فرستنده پیام

ردیف	مدجاری	Snd RACK Tout Crs	مدجدید
١	RDY	↑ - - -	SDG
٢	SDG	↓ - - -	AWT
٣	AWT	- f ↑ f	SUC
٤	AWT	- f f ↑	FAL
٥	RDY SDG AWT SUC FAL	- - - ↑	DED
٦	DED	- - - ↓	RDY

جدول ١٣ جدول گذار حالت برای گیرنده پیام

ردیف	مدجاری	Crs	Rcv	مدجدید
١	RDY	-	↑	RVG
٢	RVG	-	↓	SUC
٣	RDY RVG SUC	↑	-	DED
٤	DED	↓	-	RDY

جدول ١٤ جدول رخداد برای فرستنده پیام

مد	رخداد	
SDG	↓Snd	false
AWT	false	↑RACK when Tout= false∨ RACK=false ↑Tout when
Stm	true	false

جدول ١٥ جدول رخداد برای گیرنده پیام

مد	رخداد
RVG	↓Rcv
SndACK	true

۳-۲- ضوابط فرستنده و گیرنده در ارتباطات گروهی

در این بخش از روی جدول‌هایی که در بخش ۳-۱ به‌دست آمد (توصیف جدولی)، ضوابط فرستنده و گیرنده در ارتباطات گروهی را به‌صورت خودکار در سه بخش: (۱) ضوابط حالت، (۲) ضوابط گذار و ضوابط پاسخ تولید می‌کنیم (بخش ۲-۲ رابببنید). ضوابط حالت، حالت‌های همیشه برقرار نرم‌افزار، ضوابط گذار، گذارهای همیشه برقرار بین حالت‌های نرم‌افزار و ضوابط پاسخ، واکنش‌های همیشه لازم نرم‌افزار به رخدادهای محیطی را نشان می‌دهد. همانطور که در بخش ۲-۲ بیان کردیم، حالت‌های همیشه برقرار، حالت‌هایی را نشان می‌دهند که همیشه در این حالت‌ها، رخدادهای مشخصی تحت شرایط خاصی می‌توانند روی دهند. گذارهای همیشه برقرار، گذارهایی را نشان می‌دهند که همیشه انتقال بین حالت‌های معینی را انجام می‌دهند و پاسخ‌های همیشه لازم به رخدادهای محیطی، واکنش‌های معینی را از طریق تغییر مقدار کمیت‌های کنترلی نشان می‌دهند.

۳-۲-۱- تولید ضوابط حالت

بر طبق ضابطهٔ ۱ در بخش ۲-۲، اولین ضابطه حالت باید انحصار حالت‌های سیستم را نشان دهد. بنابراین ضوابط انحصار حالت‌ها برای فرستنده و گیرنده پیام را با توجه به جدول‌های (۸) (حالت‌های فرستنده) و (۹) (حالت‌های گیرنده) با ضوابط ۲ و ۳ بیان می‌کنیم. الگوریتم (٤) برای تولید بقیه ضوابط حالت‌های فرستنده و گیرنده پیام استفاده می‌شود. این ضوابط را با توجه به رخدادهایی که در هر حالت رخ می‌دهند و شروطی که در آن حالت‌ها برقرارند، تولید می‌کنیم. بنابراین بازای هر حالت سیستم، یک ضابطه حالت داریم. سه نمونه از ضوابط حالت یعنی ($MI_1(t)$ تا $MI_3(t)$) را برحسب منطق بی‌درنگ برای فرستنده پیام نشان می‌دهیم (ضوابط ٤ تا ٦). گزاره @val(v,-1)آخرین مقدار متغیرحالت v را نشان می‌دهد. برای مثال $MI_1(t)$ می‌گوید در هر زمانی که فرستنده پیام در حالت RDY باشد، آخرین مقدار کمیت ورودی Snd باید برابر false باشد. ضوابط کامل حالت در پیوست آورده شده است.

S_{sender} = {RDY, SDG, AWT, SUC, FAL, DED}
$S_{Receiver}$ = {RDY, RVG, SUC, DED}
$MI_0(t)_{sender}$ =
$\leftrightarrows RDY(t_1,t_2) \oplus \leftrightarrows SDG(t_1,t_2) \oplus \leftrightarrows AWT(t_1,t_2) \oplus \leftrightarrows SUC(t_1,t_2) \oplus \leftrightarrows FAL(t_1,t_2) \oplus \leftrightarrows DED(t_1,t_2)$
$MI_0(t)_{receiver}$ =
$\leftrightarrows RDY(t_1,t_2) \oplus \leftrightarrows RVG(t_1,t_2) \oplus \leftrightarrows SUC(t_1,t_2) \oplus \leftrightarrows DED(t_1,t_2)$

ضوابط ۲ و ۳ $MI_0(t)$receiver و $MI_0(t)$sender به ترتیب برای انحصار حالات فرستنده و گیرنده پیام

$MI_1(t)$ def≡ $\forall t$ if $\leftrightarrows RDY_t \rightarrow$ @val(Snd,-1) = false
$MI_2(t)$ def≡ $\forall t$ if $\leftrightarrows SDG_t \rightarrow$ @val(Snd,-1) = true
$MI_3(t)$ def≡ $\forall t$ if $\leftrightarrows AWT_t \rightarrow$ @val(Snd,-1) = false $\wedge$ @val(R_{Ack},-1) = false $\wedge$ @val(Tout,-1) = false

ضوابط ٤ تا ٦ $MI_1(t)$ تا $MI_3(t)$- سه نمونه از ضوابط حالت برای فرستنده پیام

۳-۲-۲- تولید ضوابط گذار

ضوابط گذار، ضوابط گذارهای همیشه برقرار در سیستم - یعنی گذارهایی که همیشه انتقال بین حالات معینی را انجام می‌دهند- مشخص می‌سازند. با استفاده از الگوریتم (۵)، برای هر گذار از حالت جاری به حالت جدید از جدول گذار حالت (جدول‌های ۱۲ و ۱۳) یک ضابطه برای خروج از حالت جاری و یک ضابطه برای ورود به حالت جدید برحسب منطق بی‌درنگ تولید می‌کنیم. دو نمونه از این

ضوابط با TI_{31} و TI_{32} (ضوابط ۷ و ۸) برای ردیف سوم جدول (۱۲) نشان داده شده است. فرض‌های هر جفت ضابطه را کمیت‌های ردیف سوم جدول تعیین می‌کند. به طوری که اگر مقدار کمیت، رخداد شروع کنش را نشان دهد (یعنی "↑") شرط false را برای این کمیت به قانون خروج و شرط true را به قانون ورود اضافه می‌کنیم اما اگر مقدار کمیت رخداد، خاتمه کنش را نشان دهد (یعنی ↓) شرط true را برای این کمیت به قانون خروج و شرط false را به قانون ورود اضافه می‌کنیم. بازا هر مقدار ثابت t یا f از یک کمیت، مقدار true یا false را به هر دو قانون اضافه می‌کنیم. ضوابط کامل گذار در پیوست ارائه شده است.

TI_{31} def≡ $\forall t$ if $\leftrightarrows AWT_t \wedge (\bar{\uparrow} Snd_t \wedge \bar{\uparrow} Tout_t \wedge \uparrow RAck_t) \rightarrow \nearrow AWT_{t+1}$
TI_{32} def≡ $\forall t$ if $\leftrightarrows AWT_t \wedge (\bar{\uparrow} Snd_t \wedge \bar{\uparrow} Tout_t \wedge \uparrow RAck_t) \rightarrow \searrow SUC_{t+1}$

ضوابط ۷ و ۸ گذار TI_{31} (خروج از حالت انتظار) و گذار TI_{32} (ورود به حالت موفق) برای فرستنده پیام

۳-۲-۳- تولید ضوابط پاسخ

ضوابط پاسخ، واکنش‌های همواره لازم و معین نرم‌افزار به رخدادهای محیطی را به‌وسیله تغییر کمیت‌های کنترلی نشان می‌دهد. مقدار false برای هر رخداد به این معنا است که در آن حالت، هیچگاه پاسخی نداریم. ردیف آخر هر جدول رخداد، با یک کمیت کنترلی مشخص شده که تغییر آن به‌وسیله نرم‌افزار، موجب پاسخ مناسب به محیط می‌شود. با استفاده از الگوریتم (٦)، تولید ضوابط پاسخ را برای سیستم ارتباطات گروهی نشان می‌دهیم. RI_{11} تا RI23 (ضوابط ۹ تا ۱۱)، ضوابط پاسخ را برای فرستنده پیام (جدول رخداد آن در جدول ١٤ مشخص شده) و RI_{31} (ضابطه ۱۲)، ضابطه پاسخ برای گیرنده پیام(جدول رخداد در جدول ۱۵ مشخص شده) نشان می‌دهد.

۳-۳- تعیین قیود ایمنی (دغدغه‌ها)

در این بخش، دغدغه‌های کاربران که قیود ایمنی را تشکیل می‌دهند مشخص می‌کنیم. اینها قیودی است که باید به‌وسیله نرم‌افزار(پیاده‌سازی توصیف) ارضا شوند.

RI_{11} def≡ if $\leftrightarrows SDG \wedge \downarrow Snd \rightarrow \uparrow Stm$
RI_{22} def≡ if $\leftrightarrows AWT \wedge \downarrow R_{Ack} \wedge$ @val(Tout,-1)=false$\rightarrow \downarrow Stm$
RI_{23} def≡ if $\leftrightarrows AWT \wedge \downarrow Tout \wedge$ @val(R_{Ack},-1)=false$\rightarrow \downarrow Stm$
RI_{31} def≡ if $\leftrightarrows RVG \wedge \downarrow Rcv \rightarrow \uparrow Snd_{Ack}$

ضوابط ۹ تا ۱۱ پاسخ (RI_{11} تا RI_{23}) برای فرستنده و ضابطه ۱۲- ضابطه پاسخ (RI_{31}) برای گیرنده پیام

همانطور که در بخش ۲ گفته شد، قیود ایمنی ویژگی‌هایی نیست که به‌صورت مستقیم در توصیف مسأله ظاهر شده باشند، بلکه اهدافی هستند که باید به وسیله توصیف یا پیاده‌سازی ارضا شوند. قیود ایمنی مجموعه‌ای از نبایدها برای رخدادهای مخاطره‌آمیز است. درستی‌یابی نرم‌افزار در برابر هر قید ایمنی را می‌توان بر اساس تعداد متناهی از حالت‌های اجرای نرم‌افزار بررسی کرد یعنی مشخص شود که در ضمن اجرای برنامه رخداد بدی رخ نمی‌دهد. اما به دلیل خاص‌بودن قیود ایمنی هر سیستم، تعیین این قیود برای هر سیستم باید به‌صورت خاص برای آن سیستم تعریف شود و نمی‌توان آن را به شکل عام تعیین کرد. بر طبق [۲۵] در ارتباطات گروهی باید دو ویژگی رعایت شود: (۱) ارسال یکپارچه پیام[1] و (۲) رعایت ترتیب بین پیام‌ها. برای ارضای این دو ویژگی چهار قید ایمنی در نظر می‌گیریم: قید ۱ ویژگی یکپارچگی پیام و قیود ۲ تا ٤ ویژگی‌های ترتیب پیام‌ها را نشان می‌دهند.

1. Message Delivery Integrity / Message Atomicity

قید ۱ یکپارچگی پیام با ضابطه زیر تعریف می‌شود:

قید ۱ - یکپارچگی پیام

e_0= send(server$_0$, m)

$P_1^{def}\equiv$ if $\leftrightarrows$ correct$_{t0} \wedge t_1$=@(e_0, k) $\wedge\ t_0 < t_1 \Rightarrow$ $\leftrightarrows$correct$_{t2}$: $t_1<t_2$

این ضابطه می‌گوید اگر پیامی به گروه فرستاده شود، یا تمامی اعضای گروه پیام را به‌صورت صحیح دریافت خواهند کرد یا هیچ یک از اعضا آن را دریافت نخواهند کرد (ویژگی "همه" یا "هیچ"). بنابراین اگر سیستم قبل از ارسال پیام (یعنی زمان t_0) در حالت درست قرار دارد، پس از ارسال پیام (یعنی زمان t_2) نیز باید در وضعیت درست (سازگار) قرار داشته باشد. این ویژگی در سیستم‌های توزیع‌شده مانند سیستم پایگاه‌داده توزیع‌شده مکرر[1] از ناسازگاری بین داده‌ها جلوگیری می‌کند.

قید ۲ ویژگی ترتیب سببی و همزمانی مجازی با ضابطه زیر تعریف می‌شود:

قید ۲- ویژگی ترتیب سببی و همزمانی مجازی

$Pr = \bigcup_{i=1}^{n} Pr\,ocess_i$, $E = \bigcup_{j=1}^{m} event_j$, m is a message

P_2: $\alpha \Rightarrow \beta$ def≡

@(β, j) ≤ @(α, i) + n $\wedge$ Pid (α) = Pid (β) : $\alpha, \beta \in E$

$\vee$ @(β, i) ≤ @(α, i) + n : α = Send(p,m),

β= Receive(q,m) , p,q ∈ Pr $\vee$ [$\exists\delta$: $\alpha\Rightarrow\delta \wedge \delta \Rightarrow\beta$]

این ضابطه می‌گوید دو پیام سببی باید به‌ترتیب صدور پیام برسند. اگر α، β و δ معرف سه رخداد ارسال یا دریافت پیام باشند، آنگاه رخداد β به رخداد α بستگی دارد (۱) اگر هر دو رخداد به‌وسیله یک پردازه ارسال شوند یا (۲) رخداد α معرف ارسال یک پیام به وسیله یک پردازه و β معرف رخداد دریافت همین پیام به‌وسیله پردازه دیگری باشد یا (۳) رخداد β به رخداد δ بستگی داشته باشد و رخداد δ نیز به رخداد α بستگی داشته باشد. این قید رابطه تراگذاری سببی بین رخدادها را نشان می‌دهد. با توجه به ویژگی ترتیب سببی، «ویژگی تحویل سببی پیام» و سپس «ویژگی تحویل سببی مطمئن» را تعریف می‌کنیم.

قید ۳ ویژگی تحویل سببی پیام با ضابطه زیر تعریف می‌شود:

قید ۳- ویژگی تحویل سببی پیام

$Pr = \bigcup_{i=1}^{n} Pr\,ocess_i$

P_3def≡ if [α = Send(p_0, m_0), β = Send(p_1, m_1) ,

δ = Receive(q, m_0),

σ = Receive(q, m_1)] $\wedge$ [@(β,j) ≤ @(α,i) + $n_1 \wedge \alpha \Rightarrow \beta$]

$\rightarrow$ @(σ, k) ≤ @(δ,l) + n_2 : n_1 , $n_2 > 0$,

p_0 , p_1 , q ∈ Pr , m_0 , m_1 are messages

این ضابطه بیان می‌کند که اگر دو پیام m_0 و m_1 ارسال شوند به‌طوری که m_0 قبل از m_1 فرستاده شود و m_1 بستگی سببی به m_0 داشته باشد، پس هر گیرنده‌ای که این دو پیام را دریافت کند، باید m_0 را قبل از m_1 دریافت کند.

قید ٤ ویژگی تحویل سببی مطمئن با ضابطه زیر تعریف می‌شود.

قید ٤- ویژگی تحویل سببی مطمئن

$Pr = \bigcup_{i=1}^{n} Pr\,ocess_i$

P_4def≡ if [α = Send(p_0, m_0) , β = Send(p_1, m_1),

δ = Receive(q, m_0) , σ = Receive(q, m_1)]

$\wedge$

[@(β,j) ≤ @(α,i) + $n_1 \wedge \alpha \Rightarrow \beta$] $\wedge$ @(σ,k) $\rightarrow$

@(σ,k) ≤ @(δ, l) + n_2 : n_1 , $n_2 > 0$,

p_0 , p_1 , q ∈ Pr , m_0 , m_1 are messages

این ضابطه می‌گوید اگر پیام m_0 قبل از پیام m_1 ارسال شود و m_1 بستگی سببی به m_0 داشته باشد، پس هر گیرنده‌ای که m_1 را دریافت کند، باید m_0 را قبل از m_1

1. Replicated Distributed Data Base System

دریافت کرده باشد. در قید سببی مطمئن، اگر پیام وابسته‌ای برسد، باید قبل از آن پیام مسبب رسیده باشد. در این قید، دو نتیجه باید درستی‌یابی شود، یکی وجود پیام مسبب و دیگری تقدم آن. اما در قید ٤، فقط تقدم دو پیام بررسی می‌شود. به‌طور کلی در ارتباطات گروهی ایمن، قیود ۱و٤ باید رعایت شود: $P_{Reliable} def\equiv P_1 \wedge P_4$

٤-۳- تعیین قوانین درستی‌یابی

در شکل (٤) مراحل روش تولید قوانین درستی‌یابی رفتار نرم‌افزار نشان داده شده است. اکنون با توجه به این شکل به تولید قوانین درستی‌یابی ارتباطات گروهی ایمن و مطمئن می‌پردازیم. در بخش ٤-۲ الگوریتم تولید قوانین درستی‌یابی را برای رفتار اجرایی نرم‌افزار در برابر قیود ایمنی نشان دادیم. با استفاده از این قوانین می‌توانیم از عدم انحراف رفتار اجرایی نرم‌افزار از قیود مطمئن شویم. اکنون با استفاده از این الگوریتم تعدادی از قوانین درستی‌یابی رفتار اجرایی ارتباطات گروهی را تولید می‌کنیم. فرض کنیم سرویسگر اصلی (q_0) به‌عنوان فرستنده، پیام تغییر داده‌ها را به سرویسگرهای پشتیبان به‌عنوان گیرنده ارسال می‌کند. در این صورت: (۱) آن قیود ایمنی را مشخص می‌کنیم که در فرض آنها رخداد ارسال پیام وجود دارد تا بتوانیم رفتار سیستم را در برابر تحقق نتیجه این قیود درستی‌یابی کنیم. قید ۱ یکی از این قیود است که در فرض آن رخداد ارسال پیام (یعنی $(t_1=@(send(server0,m),k))$ و درست‌بودن سیستم (تمامی سرویسگرهای گیرنده یا پشتیبان) قرار دارد. بنابراین، فرستنده در حالت SDG (ردیف اول جدول ۱۲) و گیرنده‌ها در حالت RDY (ردیف چهارم جدول ۱۳) قرار دارند. برای SDG، ضابطه MI_2 (جدول ۱ پیوست) و برای RDY ضابطه TI_{1-1} و ضابطه TI_{1-2} (جدول ۳ پیوست) تولید می‌شوند که گزاره‌های آنها را به‌عنوان قوانین درستی‌یابی انتخاب می‌کنیم (جدول ١٦).

نتیجه قید ۱ درست‌بودن سیستم را پس از ارسال پیام نشان می‌دهد که همان حالت RDY است و منجر به ارسال تصدیق (پاسخ) از طرف سرویسگرهای گیرنده می‌شود که با ضابطه RI_{31} مشخص شده (جدول ۵ پیوست) و گزاره‌های این ضابطه (یعنی قوانین درستی‌یابی) در جدول (١٦) آورده شده است.

٤- نتیجه‌گیری

در این مقاله با استفاده از هفت الگوریتم، رویکردی خودکار به نام SRG در سه مرحله ارائه و در آن

جدول ١٦ قوانین درستی‌یابی رخداد ارسال پیام

ضابطه	قانون درستی‌یابی	مورد درستی‌یابی
MI_2	⇆SDGt	فرستنده
	@val(Snd,-1)= true	فرستنده
TI_{1-1}	if ☾ ↑Sndt ☽ → ↗RDYt+1	گیرنده
TI_{1-2}	if ☾ ↑Sndt ☽ → ↘SDGt+1	گیرنده
RI_{31}	⇆RVGt	گیرنده
	if ☾ ↓Rcv ☽ → ↑SndAck	گیرنده

به تولید نظام مند قوانین درستی‌یابی رفتار در زمان اجرای نرم‌افزار پرداخته شد. مرحله اول به‌صورت دستی و مراحل دوم و سوم به‌صورت خودکار به‌وسیله هفت الگوریتم انجام شد. در مرحله نخست، محیط برحسب کمیت‌های پایشی و کنترلی مشخص شده و با این کار، تعامل محیط و سیستم تعیین و آن دو از یکدیگر متمایز شد. سپس این تعامل را با مدل‌سازی

بصری ماشین حالت نشان دادیم. در مرحله دوم، مدل بصری به یک مدل میانی که به‌صورت جدولی ارائه شد، تبدیل شد تا بتوان در مرحله سوم، قوانین درستی‌یابی را به‌صورت خودکار از آن ایجاد کرد.

اکنون به مقایسه رویکرد SRG با رویکردهای دیگر درستی‌یابی در زمان اجرای نرم‌افزار می‌پردازیم. رویکردهای دیگر را می‌توان به دو گروه: (۱) رویکردهای مبتنی بر موجودیت‌ها و فعالیت‌های نرم‌افزار و (۲) رویکردهای مبتنی بر توصیف مسأله تقسیم کرد. رویکردهای نوع اول مانند jMonitor [۲۶] فاقد نگاشت توصیف‌های انتزاعی به موجودیت‌های اجرایی است و در آن نمی‌توان با توجه به توصیف مسأله، نرم‌افزار را درستی‌یابی کرد. این رویکرد، توصیف‌های خود را با الگوهای طراحی نرم‌افزار مشخص می‌کند.

در رویکردهای گروه دوم، نیازهای ایمنی به روشی رسمی توصیف می‌شوند. در دسته اول این رویکردها، توصیف‌های رسمی، در داخل نرم‌افزار اصلی قرار می‌گیرند که انتزاع کمتری نسبت به رویکرد SRG دارند. در رویکرد SRG، توصیف‌های انتزاعی در طی دو مرحله نگاشت، به موجودیت‌های سطح کد تبدیل می‌شوند. در [۲۷] دو رویکرد RULER و LOGSCOPE و در [۲۸] Eagle برای تولید قوانین درستی‌یابی زمان اجرا معرفی شده که از زبان‌های خاصی استفاده کرده‌اند. Eagle که از زبان HAWK [۲۹] بهره می‌گیرد، به علت این که در توصیف‌های خود تعاریف سطح برنامه‌نویسی را منظور کرده، انتزاع و وضوح کمتری در توصیف نسبت به رویکرد SRG دارد. JPaX [۳۰] نیز دو نوع توصیف: (۱) برحسب موجودیت‌های کد برنامه‌های Java و (۲) برحسب گزاره‌های منطق زمانی خطی را برای نیازهای ایمنی فراهم کرده و توصیف‌های نوع (۲) را به شکل توضیحاتی در داخل برنامه‌های Java درج می‌کند.

دسته دوم رویکردهای گروه دوم انتزاع بیشتری نسبت به‌دسته اول دارند و نگاشت بین توصیف‌های انتزاعی قیود ایمنی و موجودیت‌های اجرایی را فراهم ساخته‌اند. Java-MaC [۳۱] نیازهای ایمنی را به دو زبان ویژه ارائه می‌کند: (۱) زبان اجرایی PEDL[1] برای نشان‌دادن حالت‌های نرم‌افزار و زبان توصیفی MEDL[2] برای توصیف نیازهای ایمنی. بنابراین نگاشت بین فعالیت‌های اجرایی نرم‌افزار و رخدادها و شروط انتزاعی محیطی، با تبدیل موجودیت‌هایی که برحسب دستورات PEDL بیان شده به موجودیت‌هایی که برحسب دستورات MEDL توصیف شده، انجام می‌شود. MOP [۳۲] چارچوبی را با عنوان برنامه‌نویسی پایش‌گرا[3] معرفی کرده که در آن کاربران می‌توانند یکی از گسترش‌های منطق زمانی را برای توصیف نیازهای ایمنی انتخاب کنند. کاربران با نوشتن قواعدی مشخص می‌کنند که چگونه منطق انتخابی باید به‌وسیله درستی‌یاب در زمان اجرا تفسیر شود. اجازه دادن به کاربران برای انتخاب نوع منطق زمانی که آنها می‌خواهند برای توصیف استفاده کنند، امکانی قوی است، اما تکیه بر یک منطق و زبان توصیف ویژه، پیشرفت کار را در نواحی دیگر درستی‌یابی در زمان اجرا مانند درج کد درستی‌یابی در نرم‌افزار بهتر اجازه می‌دهد.

در رویکرد SRG با الهام از [۳۳] توصیف‌ها را با تکیه بر واژگان کاربران سیستم شروع کردیم. سپس روشی را برای استخراج کمیت‌های پایشی و کنترلی از آنها ارائه کرده و سپس در دو مرحله، آنها را به

1. Primitive Event Definition Language
2. Meta-Event Definition Language
3. Monitoring Oriented Programming

موجودیت‌های سطح کد نگاشت کردیم. رویکـرد SRG –به جای نگاشت مستقیم– از توصـیف‌هـای انتزاعـی بـه توصیف‌های سـطح کـد (ماننـد دسـته دوم رویکردهـای گروه دوم) با استفاده از هفت الگوریتم، رویکردی نظامند را برای این نگاشت فراهم ساخت.

در میان ویژگی‌های رویکرد SRG، می‌توان به قابلیت استفاده برای سیستم‌های توزیع‌شده و بی‌درنگ نام برد. مثالی که در بخش ۳ بیان شد، امکان استفاده از رویکرد SRG را برای سیستم‌های توزیع‌شده نشان می‌دهد. این امکان، ناشی از اتکای رویکرد SRG به توصیف نرم‌افزار است که به‌وسیله آن می‌توان مشخص کرد که قوانین درستی‌یابی در کدام محل از سیستم توزیع‌شده باید به‌کار گرفته شود (جدول ۱۶ را ببینید). همچنین استفادهٔ رویکرد SRG از منطق بی‌درنگ در توصیف قیود ایمنی، به‌کارگیری این رویکرد را در سیستم‌های بی‌درنگ امکان‌پذیر می‌سازد.

جدول‌های پیوست

جدول ۱ ضوابط حالت برای فرستنده پیام

$MI_0(t)_{sender}$ =
$\leftrightarrows RDY_{(t1,t2)} \oplus \leftrightarrows SDG_{(t1,t2)} \oplus \leftrightarrows AWT_{(t1,t2)} \oplus$
$\leftrightarrows SUC_{(t1,t2)} \oplus \leftrightarrows FAL_{(t1,t2)} \oplus \leftrightarrows DED_{(t1,t2)}$

$MI_1(t)$ def≡ $\forall t$ if $\leftrightarrows RDY_t \rightarrow$ @val(Snd,-1) = false
$MI_2(t)$ def≡ $\forall t$ if $\leftrightarrows SDG_t \rightarrow$ @val(Snd,-1) = true
$MI_3(t)$ def≡ $\forall t$ if $\leftrightarrows AWT_t \rightarrow$ @val(Snd,-1) = false $\wedge$
@val(R_{Ack},-1) = false $\wedge$ @val(Tout,-1) = false
$MI_4(t)$ def≡ $\forall t$ if $\leftrightarrows RDY_t \rightarrow$ @val(Crs,-1) = false
$MI_5(t)$ def≡ $\forall t$ if $\leftrightarrows SDG_t \rightarrow$ @val(Crs,-1) = false
$MI_6(t)$ def≡ $\forall t$ if $\leftrightarrows AWT_t \rightarrow$ @val(Crs,-1) = false
$MI_7(t)$ def≡ $\forall t$ if $\leftrightarrows SUC_t \rightarrow$ @val(Crs,-1) = false
$MI_8(t)$ def≡ $\forall t$ if $\leftrightarrows FAL_t \rightarrow$ @val(Crs,-1) = false
$MI_9(t)$ def≡ $\forall t$ if $\leftrightarrows DED_t \rightarrow$ @val(Crs,-1) = true

جدول ۲ ضوابط حالت برای گیرنده پیام

$MI_0(t)_{receiver}$ =
$\leftrightarrows RDY_{(t1,t2)} \oplus \leftrightarrows RVG_{(t1,t2)} \oplus \leftrightarrows SUC_{(t1,t2)} \oplus \leftrightarrows DED_{(t1,t2)}$

$MI_1(t)$ def≡ $\forall t$ if $\leftrightarrows RDY_t \rightarrow$ @val(Rcv,-1) = false
$MI_2(t)$ def≡ $\forall t$ if $\leftrightarrows RVG_t \rightarrow$ @val(Crs,-1) = false
$MI_3(t)$ def≡ $\forall t$ if $\leftrightarrows SUC_t \rightarrow$ @val(Crs,-1) = false
$MI_4(t)$ def≡ $\forall t$ if $\leftrightarrows DED_t \rightarrow$ @val(CRS,-1) = true

جدول ۳ ضوابط گذار برای فرستنده پیام

$TI_{1\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows RDY_t \wedge \uparrow Snd_t) \rightarrow \nearrow RDY_{t+1}$
$TI_{1\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows RDY_t \wedge \uparrow Snd_t) \rightarrow \searrow SDG_{t+1}$
$TI_{2\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows SDG_t \wedge \downarrow Snd_t) \rightarrow \nearrow SDG_{t+1}$
$TI_{2\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows SDG_t \wedge \downarrow Snd_t) \rightarrow \searrow AWT_{t+1}$
$TI_{3\text{-}1}$ def≡ $\forall t$ if $\leftrightarrows AWT_t \wedge (\bar{\dagger} Snd_t \wedge \bar{\dagger} Tout_t \wedge \uparrow R_{Ack\,t}) \rightarrow \nearrow AWT_{t+1}$
$TI_{3\text{-}2}$ def≡ $\forall t$ if $\leftrightarrows AWT_t \wedge (\bar{\dagger} Snd_t \wedge \bar{\dagger} Tout_t \wedge \uparrow R_{Ack\,t}) \rightarrow \searrow SUC_{t+1}$
$TI_{4\text{-}1}$ def≡ $\forall t$ if $\leftrightarrows AWT_t \wedge (\bar{\dagger} Snd_t \wedge \bar{\dagger} R_{Ack\,t} \wedge \uparrow Tout_t) \rightarrow \nearrow AWT_{t+1}$
$TI_{4\text{-}2}$ def≡ $\forall t$ if $\leftrightarrows AWT_t \wedge (\bar{\dagger} Snd_t \wedge \bar{\dagger} R_{Ack\,t} \wedge \uparrow Tout_t) \rightarrow \searrow FAL_{t+1}$
$TI_{5\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows RDY_t \wedge \uparrow Crs_t) \rightarrow \nearrow RDY_{t+1}$
$TI_{5\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows RDY_t \wedge \uparrow Crs_t) \rightarrow \searrow DED_{t+1}$
$TI_{6\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows SDG_t \wedge \uparrow Crs_t) \rightarrow \nearrow SDG_{t+1}$
$TI_{6\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows SDG_t \wedge \uparrow Crs_t) \rightarrow \searrow DED_{t+1}$
$TI_{7\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows AWT_t \wedge \uparrow Crs_t) \rightarrow \nearrow AWT_{t+1}$
$TI_{7\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows AWT_t \wedge \uparrow Crs_t) \rightarrow \searrow DED_{t+1}$
$TI_{8\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows SUC_t \wedge \uparrow Crs_t) \rightarrow \nearrow SUC_{t+1}$
$TI_{8\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows SUC_t \wedge \uparrow Crs_t) \rightarrow \searrow DED_{t+1}$
$TI_{9\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows FAL_t \wedge \uparrow Crs_t) \rightarrow \nearrow FAL_{t+1}$
$TI_{9\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows FAL_t \wedge \uparrow Crs_t) \rightarrow \searrow DED_{t+1}$
$TI_{10\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows DED_t \wedge \downarrow Crs_t) \rightarrow \nearrow DED_{t+1}$
$TI_{10\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows DED_t \wedge \downarrow Crs_t) \rightarrow \searrow RDY_{t+1}$

جدول ٤ ضوابط گذار برای گیرنده پیام

$TI_{1\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows RDY_t \wedge \uparrow Rcv_t) \rightarrow \nearrow RDY_{t+1}$
$TI_{1\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows RDY_t \wedge \uparrow Rcv_t) \rightarrow \searrow RVG_{t+1}$
$TI_{2\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows RVG_t \wedge \downarrow Rcv_t) \rightarrow \nearrow RVG_{t+1}$
$TI_{2\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows RVG_t \wedge \downarrow Rcv_t) \rightarrow \searrow SUC_{t+1}$
$TI_{3\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows RDY_t \wedge \uparrow Crs_t) \rightarrow \nearrow RDY_{t+1}$
$TI_{3\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows RDY_t \wedge \uparrow Crs_t) \rightarrow \searrow DED_{t+1}$
$TI_{4\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows RCV_t \wedge \uparrow Crs_t) \rightarrow \nearrow RCV_{t+1}$
$TI_{4\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows RCV_t \wedge \uparrow Crs_t) \rightarrow \searrow DED_{t+1}$
$TI_{5\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows SUC_t \wedge \uparrow Crs_t) \rightarrow \nearrow SUC_{t+1}$
$TI_{5\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows SUC_t \wedge \uparrow Crs_t) \rightarrow \searrow DED_{t+1}$
$TI_{6\text{-}1}$ def≡ $\forall t$ if $(\leftrightarrows DED_t \wedge \downarrow Crs_t) \rightarrow \nearrow DED_{t+1}$
$TI_{6\text{-}2}$ def≡ $\forall t$ if $(\leftrightarrows DED_t \wedge \downarrow Crs_t) \rightarrow \searrow RDY_{t+1}$

جدول ۵ ضوابط پاسخ (RI_{11} تا RI_{23}) برای فرستنده و ضابطه پاسخ (RI_{31}) برای گیرنده پیام

RI_{11} def≡ if $\leftrightarrows SDG \wedge \downarrow Snd \rightarrow \uparrow Stm$
RI_{22} def≡ if $\leftrightarrows AWT \wedge \downarrow R_{Ack} \wedge$ @val(Tout, 1)= false $\rightarrow \downarrow Stm$
RI_{23} def≡ if $\leftrightarrows AWT \wedge \downarrow Tout \wedge$ @val(R_{Ack},-1)=false $\rightarrow \downarrow Stm$
RI_{31} def≡ if $\leftrightarrows RVG \wedge \downarrow Rcv \rightarrow \uparrow Snd_{Ack}$

۵- منابع

[1] Gahl D.J., Dijkstra E.J. and Hoare C.A.R., "Notes on Structured Programming", Academic Press London, pp.182, 1972.

[2] Yong S. H. and Horwitz S., "Protecting C Programs from Attacks via Invalid Pointer Dereferences" In Proceedings of the 9th ESEC/SIGSOFT ACM on Foundations of Software Engineering: 307-316, 2003.

[1] Microsoft Security Bulletin MS04-028, Version 3.0, 2004.

[2] Gates A.Q. and Teller P.J., "DynaMICs: An Automated and Independent Software-Fault Detection Approach", In Proceedings of Fourth International High-Assurance System Engineering Symposium, pp. 11-19, 1999.

[3] Butler R.W. and Finelli G.B., "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software", IEEE Transactions on Software Engineering, (19):3-12, 1993.

[4] Littlewood B. and Stringini L., "Validation of Ultrahigh Dependability for Software-based Systems", Communications of the ACM, 11(36):69-80, 1993.

[5] Borger E. and Stark R., "Abstract State Machine, A Method for High-Level System Design and Analysis", Springer-Verlag, 2003.

[6] Damm W. and Harel D., "LSCs: Breathing Life into Message Sequence Charts", Formal Methods in System Design, 19(1):45-80, 2001.

[7] Bengtsson J. and Yi W., "Timed Automata: Semantics, Algorithms and Tools", In Lecture Notes on Concurrency and Petri Nets., Reisig W. and Rozenberg G. (Eds.), LNCS 3098, Springer-Verlag, 2004.

[8] Milner R., "Communication and Concurrency", Prentice-Hall, 1989.

[9] David R. and Alla H., "Discrete, Continuous, and Hybrid Petri Nets", Springer-Verlag, 2005.

[10] Harel D. and Politi M., "Modeling Reactive Systems with Statecharts", McGraw-Hill, 1998.

[11] Drusinsky D., "Modeling and Verification Using UML Statecharts", Elsevier, 2006.

[12] Qiao Y., Zhong K., Wang H. and Li X., "Database Theory, Technology, and Applications: Developing Event-Condition-Action Rules in Real-time Active Database", In Proceedings of the ACM symposium on Applied computing SAC '07, 2007.

[13] Heitmeyer C., Archer M., Bharadwaj R. and Jeffords R., "Tools for Constructing Requirements Specifications: The SCR Toolset at the Age of Ten", International Journal of Computer Systems Science and Engineering, 20(1): 19-35, 2005.

[14] Robinson W. N., Pawlowski S. D., and Volkov V., "Requirements Interaction Management", In ACM Computing Surveys, 35(2): 132–190, 2003.

[15] Emerson E. A., "Temporal and Modal Logic", Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, Elsevier Science, pp. 995-1072, 1995.

[16] Cheng A. M. K., "Real-Time Systems, Scheduling, Analysis, and Verification", John Wiley & Sons, 2002.

[17] Mueller E. T., "Commonsense Reasoning", In Handbook of Knowledge Representation, Hermelen F.V., Lifschitz V., and Porter, B. (Eds.), Elsevier, 2006.

[18] Bellini P., Mattolini R. and Nesi P., "Temporal Logics for Real-Time System Specification", ACM Computing Surveys, 32(1): 12-42, 2000.

[19] Mattolini R. and Nesi P., "An Interval Logic for Real-time System Specification", IEEE Transactions on Software Engineering, 27(3): 208-227, 2001.

[20] Artho, C. et al., "Combining Test Case Generation and Runtime Verification", Journal of Theoretical Computer Science, Elsevier, 336 (2-3):209-234, 2005.

[21] Leucker, M. and Schallhart, C., "A Brief Account of Runtime Verification", Journal of Logic and Algebraic Programming, Elsevier, 78(5):293-303, 2009.

[22] Tanenbaum A. S. and. Steen M.V., "Distributed Systems: Principles and Paradigms", Prentice-Hall, 2007.

[23] Vitenberg R., Keidar I., Chockler G.V. and Dolev D., "Group Communication Specifications: A Comprehensive Survey", In Acm Computing Survey, 33(4): 427-469, 2001.

[24] Karaorman M. and Freeman J., "jMonitor: Java Runtime Event Specification and Monitoring Library", In Proceedings of the 4th International Workshop on Runtime Verification, Elsevier journal of ENTCS, 113: 181-200, 2005.

[25] Barringer H., Havelund K., Rydeheard D. and Groce. A., "Rule Systems for Runtime Verification: A Short Tutorial", Lecture Notes in Computer Science, Springer, pp. 1-24, 2009.

[26] Barringer H., Goldberg A., Havelund K. and Sen K., "Rule-Based Runtime Verification", Journal of Verfication, Model Checking and Abstract Interpretation, Springer, Vol. 2937, 2004.

[27] d'Amiron M. and Havelund K., "Event-Based Runtime Verification of Java Programs", In Proceedings of the 5th Workshop of ICSE(International Conference on Software Engineering) on Dynamic Analysis, 2005.

[28] Havelund K. and rosu G., "An Overview of the Runtime Verification Tool Java Path Explorer", Formal Methods in System Design, 24(2): 189-215, March 2004.

[29] Viswanatha M. and Kim M., "Foundations for the Run-Time Monitoring of Reactive Systems-Fundamentals of the MaC Language", Lecture Notes in Computer Science, Springer, pp. 543-556, 2005.

[30] Chen F. and Rosu G., "Mop: an efficient and generic runtime verification framework", In Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications, pp. 569-588, 2007.

[31] Peters D. K., Parnas D. L., "Requirements-based Monitors for Real-Time Systems", IEEE Transaction on Software Engineering, 28(2): 146-158, 2002.